Understanding Convolutional Neural Networks

Ruth Fong

St. John's College
University of Oxford

A thesis submitted for the degree of Doctor of Philosophy

Michaelmas 2020

Abstract

In the past decade, deep learning has fueled a number of exciting developments in artificial intelligence (AI). However, as deep learning is increasingly being applied to high-impact domains, like medical diagnosis or autonomous driving, the impact of its failures also increases. Because of their high complexity (*i.e.* they are typically composed of millions of parameters), deep learning models are difficult to interpret. Thus, there is a great need for tools that help us understand how such models make their decisions. In this thesis, we introduce several methods for understanding convolutional neural networks (CNNs), the class of deep learning models that is typically applied to visual data — *i.e.* images and videos. Our techniques span three approaches to understanding a model: 1. describing the relationship between its inputs and outputs; 2. characterizing correlations between a model's inputs and its *internal* representation; and 3. using visualization tools to easily and efficiently *explore* aspects of a model.

First, we tackle the attribution problem of identifying the parts of a model's input (*i.e.* image regions) that are most responsible for its output decision. We present two techniques — *meaningful perturbations* and *extremal perturbations* – which work by perturbing the input image and learning the regions that when edited out, most affect the model's prediction. Second, we seek to understand how semantic concepts, from different kinds of textures to various kinds of objects, are recognized by network parameters (*a.k.a.* neurons). We introduce *Net2Vec*, a novel paradigm that reveals how combinations of internal neurons encode specific concepts. Lastly, similar to how a stethoscope is used to explore the internal behavior of different parts of the body, we introduce a novel visualization technique — *interactive similarity overlays* — that allows an AI researcher or developer to quickly and easily explore the internal representation of a model. Together, these methods enable us to scientifically understand the external behavior of CNNs as well as their inner workings.

Understanding Convolutional Neural Networks

Ruth Fong
St. John's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2020

Declaration

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, described my own research.

Ruth Fong, St. John's College

"For now we see only a reflection as in a mirror; then we shall see face to face. Now I know in part; then I shall know fully, even as I am fully known."

– 1 Corinthians 13:12

To frontline workers who kept the world afloat during the COVID-19 pandemic. Thank you for your selfless service; I hope to remember and honor the jobs and people that are truly essential to every community.

—

To civil rights activists, past and present. Thank you for your persistent activism to seek the good of those on the margins; I hope to follow your example of seeking justice and defending the oppressed.

—

To our parents, in life and love, who sacrificed much to give us their best.

To the ones with us today, Tung Fong, Chin-lin Fong, and Barbara Zhang. Thank you for your continual support of our studies abroad. We look forward to coming home soon, Lord willing.

In memory of the one who has gone before us, Shoucheng Zhang. We remember your zest for life and learning and continue to carry your example of unbridled enthusiasm and child-like love for learning new things.

Acknowledgements

This thesis was written in incredible circumstances that challenged the world. The first is the COVID-19 pandemic; the second is renewed, global Black Lives Matter protests. Breonna Taylor, Ahmaud Arbery, and George Floyd are only a few names of those who died by racism during this period, which also include the disproportionate number of BAME individuals who have died of COVID-19. To write up in these times is a great privilege; thus, I acknowledge the deep racial and economic inequalities exposed by both circumstances.

Regarding this thesis and the PhD it represents, I would first like to thank my advisor, Andrea Vedaldi, for his constant support and encouragement. Beyond his professional mentorship, his consistent willingness to personally support me and my family through unexpected circumstances has been greatly appreciated. I am also grateful for the support of the Rhodes Trust and Open Philanthropy and for the people who turned scholarships into supportive communities, particularly Daniel Dewey, Mary Eaton, Nadiya Figueroa, and Catherine Olsson.

I would also like to thank my examiners, Andrew Zisserman and Antonio Torralba, for their leadership in the field. I am grateful to Been Kim and Chris Olah, whose encouragement and work on deep understanding continues to inspire me. I am also indebted to former mentors who first sparked my interest in computer science and research: Wendy Gall, Walter Scheirer, and David Cox.

I am incredibly grateful to my collaborators, who have shown me numerous ways to do research beautifully. In particular, I am thankful to Mandela Patrick and Kurtis David for graciously bearing with me as I learned how to be a mentor and for bringing fresh ideas and excitement to every discussion, and to Sylvestre-Alvise Rebuffi and Xu Ji for enriching our collaboration with their unique research styles and skills. I have also been fortunate to spend two lovely summers at Google Research in Zürich and Facebook AI Research in London and am thankful to Vitto Ferrari, Rodrigo Benenson, and Andrea Vedaldi for their support at these internships.

I am also thankful to the members of the Visual Geometry Group (VGG) and the countless conversations over lunch and tea. I am especially grateful to those who contributed greatly to VGG's social activities: Fatma Güney, Gül Varol, and Shangzhe Wu. I am also thankful for the essential workers of VGG: Maddy Crudge, Jenny Hu, Ashish Thandavan, and Cassandra Warren. I am particularly grateful

to Samuel Albanie for altruistic debugging sessions, and to Karel Lenc for deftly managing the clusters alongside his own studies.

I would also like to thank several colleagues who have greatly enriched my PhD journey. First, I am incredibly grateful to the women of VGG, who have provided a lovely support network and have helped make the lab a more inclusive place for all. I am also thankful for those in my cohort who have enhanced the journey with much laughter and conversation: Sebastien Ehrhardt, Arsha Nagrani, Sylvestre-Alvise Rebuffi, and Olivia Wiles. I am particularly grateful to Sophia Almut Koepke, Erika Lu, and Aravindh Mahendran, for walking with me and Brian through the difficult days; their friendship in and out of the lab have been invaluable and life-giving.

I am also thankful for the miniature bodies of Christ we have been blessed to be a part of over the past five years: Rhodes Christian Fellowship, St. Andrews, International Protestant Church (Zürich), and Emmanuel. We are particularly grateful to a few people: Jonathan and Melissa Grant-Peters and Kirsten Mackerras, for leading a small group with us and for their continued friendship; the leaders of the culture connect group — Lucy Norris, Faith Oyegbile, the Baches, Mudaus, and van Bever Donkers — for their faithful servant leadership; and the Bache party of four — Tyron, Kelly, Quinton, and Amor — and Joel and Tash Robertson, for their quarantine support and prayers that fueled this thesis' writing.

There are too many others to name who have supported us these past few years. I am particularly thankful for Anisha Gururaj, Sean Lau, and Sarah Yang, for friendships that started in Oxford and have continued to grow over the years. I am also thankful for "Da View," the village that continues to raise me: Meredith Arra, Rosa Huang, Chloe Reichel, and Carrie Tian. I am also grateful to Vivian Chan Li, Brian and Shella Gifford, and Jenny Shih, for their spiritual friendship and prayers. Lastly, this thesis was significantly improved thanks to the sharp eyes and suggestions of Sophia Almut Koepke, Sean Lau, Carrie Tian, and Brian Zhang.

We are also incredibly thankful for our families' support. Living in the U.K. has given us a glimpse into the sacrifices our parents made for us when they moved to the U.S., which have made all our degrees possible. We are also thankful for the adult friendships with our siblings — Tiffany, Jonathan, and Stephanie — and their partners — Stetson and Charlie — that continue to encourage us.

I would like to especially thank my *ezer*, Brian Zhang. This PhD simply would not have been possible without his sacrificial love, manifest in countless homecooked meals, many hours proofreading drafts and fixing graphs, and a deep spiritual friendship that points me to Christ every day. Words cannot express my gratitude to my husband, and every success is shared with him.

Lastly, I am eternally grateful to my Maker. The more I study and create, the more I am in awe of the Creator. His lavish love daily fills me, redeems my shortcomings, and empowers me to live freely. May I live every day for His will to be done and not my own. *Solo Deo Gloria.*

Abstract

In the past decade, deep learning has fueled a number of exciting developments in artificial intelligence (AI). However, as deep learning is increasingly being applied to high-impact domains, like medical diagnosis or autonomous driving, the impact of its failures also increases. Because of their high complexity (*i.e.* they are typically composed of millions of parameters), deep learning models are difficult to interpret. Thus, there is a great need for tools that help us understand how such models make their decisions. In this thesis, we introduce several methods for understanding convolutional neural networks (CNNs), the class of deep learning models that is typically applied to visual data — *i.e.* images and videos. Our techniques span three approaches to understanding a model: 1. describing the relationship between its inputs and outputs; 2. characterizing correlations between a model's inputs and its *internal* representation; and 3. using visualization tools to easily and efficiently *explore* aspects of a model.

First, we tackle the attribution problem of identifying the parts of a model's input (*i.e.* image regions) that are most responsible for its output decision. We present two techniques — *meaningful perturbations* and *extremal perturbations* – which work by perturbing the input image and learning the regions that when edited out, most affect the model's prediction. Second, we seek to understand how semantic concepts, from different kinds of textures to various kinds of objects, are recognized by network parameters (*a.k.a.* neurons). We introduce *Net2Vec*, a novel paradigm that reveals how combinations of internal neurons encode specific concepts. Lastly, similar to how a stethoscope is used to explore the internal behavior of different parts of the body, we introduce a novel visualization technique — *interactive similarity overlays* — that allows an AI researcher or developer to quickly and easily explore the internal representation of a model. Together, these methods enable us to scientifically understand the external behavior of CNNs as well as their inner workings.

Publications

The following works contain the central findings of this thesis:

- **Ruth Fong**[§] and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *ICCV*, 2017. (Fong et al., 2017)
- **Ruth Fong** and Andrea Vedaldi. Net2Vec: Quantifying and Explaining how Concepts are Encoded by Filters in Deep Neural Networks. *CVPR*, 2018. (Fong et al., 2018b)
- **Ruth Fong**[*], Mandela Patrick[*], and Andrea Vedaldi. Understanding Deep Networks via Extremal Perturbations and Smooth Masks. *ICCV*, 2019. (Fong et al., 2019a)
- **Ruth Fong**, Alexander Mordvintsev, Andrea Vedaldi, and Chris Olah.[†] Interactive Similarity Overlays. *In preparation*. (Fong et al., in prep)

The following publications, technical reports, and manuscripts were produced in related areas of research during the course of my PhD:

- **Ruth Fong**[§], Walter Scheirer, and David Cox. Using Human Brain Activity to Guide Machine Learning. *Scientific Reports*, 2018. 1-10. (Fong et al., 2018a)
- **Ruth Fong** and Andrea Vedaldi.[1] Explanations for Attributing Deep Neural Network Predictions. *Explainable AI: Interpreting, Explaining, and Visualizing Deep Learning*. Springer Cham, 2019. 149-167. (Fong et al., 2019b)
- **Ruth Fong**[§] and Andrea Vedaldi. Occlusions for Effective Data Augmentation in Image Classification. *ICCV Workshop on Interpreting and Explaining Visual Artificial Intelligence Models*, 2019. (Fong et al., 2019c)
- Sylvestre-Alvise Rebuffi[*], **Ruth Fong**[*], Xu Ji[*], and Andrea Vedaldi. There and Back Again: Revisiting Backpropagation Saliency Methods. *CVPR*, 2020. (Rebuffi et al., 2020)

[*] denotes equal contribution. [†] denotes author order not finalized. [§] denotes being published under the name "Ruth C. Fong." For consistency, we omit the middle initial in this thesis.

[1] Fong and Vedaldi, Springer Cham 2019 is a book chapter that expands upon Fong and Vedaldi, ICCV 2017.

- Mandela Patrick*, Yuki Asano*, Polina Kuznetsova, **Ruth Fong**, João Henriques, Geoffrey Zweig, and Andrea Vedaldi. Multi-modal Self-Supervision from Generalized Data Transformations. *arXiv*, 2020. (Patrick et al., 2020)

- Miles Brundage*, Shahar Avin*, Jasmine Wang*, Haydn Belfield*, Gretchen Krueger*, Gillian Hadfield, Heidy Khlaaf, Jingying Yang, Helen Toner, **Ruth Fong**, et al. Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims. *arXiv*, 2020. (Brundage et al., 2020)

- Diego Marcos, **Ruth Fong**, Sylvain Lobry, Rémi Flamary, Nicolas Courty, and Devis Tuia.† Contextual Semantic Interpretability. *ACCV*, 2020. (Marcos et al., 2020)

- Han Peng, **Ruth Fong**, Gwenaëlle Douaud, Christian Beckman, Andrea Vedaldi, and Stephen Smith.† Region-Aligned Prediction: Population-level interpretation of deep convolutional network layers in neuroimaging. *Under review.* (Peng et al., under review)

- Iro Laina, **Ruth Fong**, Andrea Vedaldi.† Quantifying Learnability and Describability of Visual Concepts Emerging in Representation Learning. *NeurIPS*, 2020. (Laina et al., 2020)

- Kurtis David, **Ruth Fong**, Qiang Liu.† Debiasing Convolutional Neural Networks via Meta Orthogonalization. *NeurIPS Workshop on Algorithmic Fairness through the Lens of Causality and Interpretability*, 2020. (David et al., 2020)

Contents

Appendices

List of Figures

1 Introduction

Contents

1.1 Motivation

In the past decade, an explosion of exciting developments in artificial intelligence (AI) has been fueled by deep learning.[1]Notable examples include advances in medical imaging, machine translation,[2] and self-driving cars. However, as deep learning is increasingly being applied to applications with a high impact on society, the impact of its failures also increases. For instance, if there is a medical misdiagnosis or a car crash, such tools could offer an explanation and suggest a fix for the failure. Thus, there is a great need for tools that help us understand what AI systems are actually doing.

[1]Deep learning refers to the use of an artificial neural network composed of many layers.

[2]Machine translation is the automated process of translating from one human language to another.

We would also want to ensure that protected features such as race or gender are not being used to make decisions. In the case of early machine translation work, researchers noticed quickly that the learned representations reflected the biases that occur naturally in human data. For example, "doctor" became a gendered concept that was more closely aligned to male pronouns (Bolukbasi et al., 2016; Johnson, 2018; Johnson, 2020).

Given the high complexity of modern AI systems,[3] they are not easily interpretable (*i.e.* easy to understand). Currently, AI systems are typically evaluated based on their performance on well-curated datasets.[4] While performance metrics provide a concise signal of whether a model behaves as expected, they fail to explain a model's decision-making process. Thus, such *interpretability tools* would help us develop trust in these automated systems as well as understand their failure modes.

1.2 Research themes, contributions, and outline

In this section, we summarize the research contributions and structure of this thesis. We first outline the scope of this thesis. Then, we introduce a useful analogy which we use to explain the research themes that comprise this thesis.

1.2.1 Scope of thesis

In this thesis, we are concerned with understanding a particular kind of machine learning model: *convolutional neural networks* (CNNs). This class of models typically takes as input *visual data* (*e.g.* images and videos). Furthermore, we primarily consider a CNN model that performs *object classification* on images.[5,6] In this setting, a model takes as input an image and must learn to predict the most dominant object in the image from a set of possibilities (*e.g.* specific breeds and species of animals, things commonly found in a kitchen, etc.). That said, our techniques can easily be adapted to other kinds of image classifiers and extended to other kinds of neural networks (see chapter 7).

[3] Deep learning models rely on millions of parameters.

[4] An example of such a dataset is ImageNet (Russakovsky et al., 2015), a large-scale dataset of over 14 million images that is most commonly used for object classification, as it provides image-level labels for 1000 mutually-exclusive object classes (*e.g.* different breeds of animals).

[5] See these notes on "Image Classification."

[6] See section 7.4.1 for why we primarily study object classification.

1.2.2 An analogy: the human body

Consider another high-functioning, complex system: the human body. Without work by the medical and scientific communities, our understanding of the human body would be quite limited. However, with the development of modern science and medicine, patients now expect their doctors to be able to diagnose various ailments. This is made possible by a few kinds of tools used by medical professionals and researchers.

The first are physical examinations in which a healthcare professional examines an individual based on their symptoms and performs a few simple, minimally invasive tests (*e.g.* reflex tests, eye and hearing exams, allergen tests). These typically synthesize relevant information based on an observation of symptoms and responses to external stimuli (*e.g.* knee jerk reactions, improved eyesight with prescription lenses, allergic responses).

The second are more advanced medical tests and procedures (*e.g.* medical imaging procedures, endoscopic exams). These vary in how invasive they are but often provide richer information about the internal state of a patient.

The third are medical devices and equipment that enable medical workers to easily and efficiently explore potential medical problems (*e.g.* stethoscopes, X-ray machines). Often, these are readily available, easy to use, and provide rapid results.

This categorization maps neatly and respectively onto the three research themes that are highlighted in this dissertation and can be described as answering the following questions about an *image classifier*:[7]

1. Observing the external: What is the image classifier "looking at" in an input image to make its decision?

2. Understanding the internal: How does it encode semantic information, like textures and objects?

3. Using exploratory tools: How can we easily explore its internal representation?

1.2.3 Theme 1: Observing the external

Analog to medical exams. Like a doctor who performs a medical examination and observes their patient's bodily response to different stimuli (*e.g.* a skin-prick *allergy test*, in which a doctor exposes a patient to a series of allergens and observes which ones cause a reaction), AI researchers and practitioners can similarly gain

[7] *i.e.* a CNN that performs object classification.

	Medicine	Artificial Intelligence (AI)
Practitioners	healthcare workers (*e.g.* doctors, nurses, etc.)	software developers (*e.g.* industry, open-source, etc.)
Researchers	medical researchers (*e.g.* basic science, clinical work, etc.)	AI researchers (*e.g.* university labs, industry labs)
Corporations	pharmaceuticals, biotech industry	digital technology companies
Regulators	MHRA in the UK, FDA in the US	TBD
Observing external symptoms	physical examinations (*e.g.* allergen test)	"black-box" interpretability (*e.g.* attribution heatmaps)
Understanding internal condition	advanced medical tests and procedures (*e.g.* molecular imaging)	"white-box" interpretability (*e.g.* concept vectors)
Using exploratory tools to form hypotheses	medical devices and equipment (*e.g.* stethoscope)	interactive visualizations and interfaces (*e.g.* interactive similarity overlays)

Figure 1.1: Medical analogy.

insight about a model's behavior by synthesizing observations of a model's response to a variety of stimuli.

Attribution. Consider how one might explain *why* an image classifier predicted "golden retriever" as the most dominant object for a particular image.

One way to explain such a prediction is to highlight the parts of the input (*i.e.* regions in the image) that are responsible for the model's "golden retriever" prediction via a heatmap visualization. This is formally known as the *attribution* problem, because it is concerned with attributing the parts of the input that are responsible for the model's output.

A natural way to tackle this problem is to learn how to edit the input image such that only the essential parts of the image are preserved, yet the image classifier still correctly predicts "Golden retriever." Or conversely, only the essential parts are deleted, yet the image classifier now makes an incorrect prediction.

1.2.3.1 Meaningful perturbations

In chapter 3 (Fong et al., 2017), we introduce a novel algorithm — **meaningful perturbations** — that learns this essential set of image regions that, when edited, either preserves or destroys the model's predictive ability for a given object class

Figure 1.2: Meaningful perturbations (Fong et al., 2017). In this example, a mask is learned (right) and used to edit (*i.e.* blur) an image (middle) so that the image classifier's "flute" prediction is changed from 99% confidence to less than 1% confidence (left: original image).

(*e.g.* "golden retriever"). We do this by learning[8] an *minimal* and *smooth* mask that is used to perturb (*i.e.* blur) specific regions in the input image.[9,10]

A *minimal* mask is preferred because one possible mask is a one that blurs the whole image. However, that would be uninformative, as it does not identify what parts of the image are important to the model. Thus, we want to identify the minimal, essential set of image regions that are critical to a prediction.

A *smooth* mask is also preferred because another possible mask is one that contains a highly unnatural (*i.e.* unsmooth) pattern that "tricks" the model into thinking a different object is present (*e.g.* by "drawing" another object).[11] This is also uninformative, because we are interested in learning what parts of the image contain positive evidence for the original prediction.

In this work, we also outline a formal framework for conceptualizing explanations as meta-predictors that predict the behavior of the model for certain stimuli. In this "explanations as meta-predictors" framework, our learned masks can be seen as predictions of the regions that most affect a model's decision-making process (*e.g.* if deleted, the model would make an incorrect prediction).

[8]Formally, we learn the values of the mask via optimization using backpropagation; see appendix B.1.4 for a primer on optimization and backpropagation for deep learning.

[9]The values in the mask are constrained to be between 0 and 1, where 1 denotes fully perturbing a pixel and 0 denotes leaving a pixel as is.

[10]We replace image regions with a blurred versions of themselves according to the mask. The mask is the same size as the input image; thus, the amount of blur to apply is denoted by the corresponding value in the mask for a given pixel (*e.g.* 1 denotes strongly blurring a pixel, 0 denotes applying no blur). We also explore other kinds of perturbations such as random noise and a constant value (*e.g.* the mean pixel value).

[11]This is closely related to work on adversarial examples (Szegedy et al., 2014; I. J. Goodfellow et al., 2015), which take real images and make small, inperceptible edits to them such that they are then misclassified by a model.

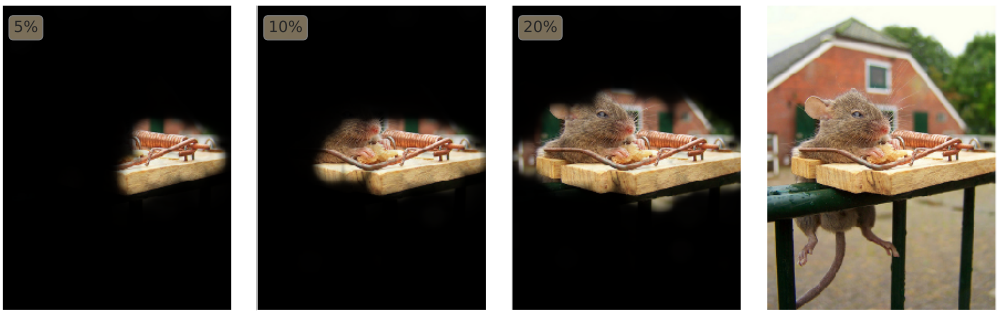


Figure 1.3: Extremal perturbations (Fong et al., 2019a). In this example, several area-constrained masks are learned (areas in inset boxes). When these masks are used to blur the original image (right), an image classifier still predicts "mousetrap" with high confidence, even though it only sees a small portion of the image. Together, these masks rank image regions in decreasing importance to the classifier (*e.g.* the mousetrap's coil is most important). For clarity, the blurred regions are blacked out.

1.2.3.2 Extremal perturbations

In chapter 4 (Fong et al., 2019a), we introduce another algorithm, named **extremal perturbations**, that improves on our previous method in several ways.

First, we introduce the ability to learn *area-constrained binary* masks[12] as explanations, that is, our method is able to find the $X\%$ (*e.g.* $5\%, 10\%$, etc.) of the image that is most responsible for the model's decision.

Second, our extremal perturbations method also leverages a novel mechanism for ensuring *smooth* masks.[13] In our previous work, we had to trade off the attribution quality of the explanation (*e.g.* how well it preserved or destroyed predictive ability) with its interpretability (*e.g.* how minimal and smooth it was).[14] In our new work, we eliminate this tradeoff by ensuring by design that our masks are area-constrained and smooth; thus, we are able to find the best masks based purely on attribution quality.

Third, we extend our work from attributing spatial regions in the input image to attributing neurons inside the CNN. We do this by we learning the essential set of neurons that, when preserved, similarly preserve the model's prediction.

Our channel attribution method is somewhat analogous to optogentic experiments with mice. In these experiments, neurons in a mouse's brain have been genetically modified and connected to an external sensor such that they can be

[12] A *binary* mask is constrained to only contain 0s and 1s.

[13] This is done introducing a parametric family of smooth masks.

[14] This is because we learned mask by minimizing a combination of loss terms: $\min \mathcal{L}_{\text{delete}} + \mathcal{L}_{\text{smooth}} + \mathcal{L}_{\text{minimal}}$. This optimization pits the different terms against each other: for instance, a mask that is very minimal (*i.e.* low $\mathcal{L}_{\text{minimal}}$) may not be strong enough to destroy a model's predictive ability (*i.e.* high $\mathcal{L}_{\text{delete}}$).

controlled by the presence or absence of light.[15] In our work, we similarly control the "expression" of artificial neurons in a CNN using a mask that we learn and use to perturb neurons.[16]

Together, these two chapters highlight our work on learning instance-specific perturbations as explanations that attribute which parts of the input image (or which intermediate neurons) are responsible for a model's prediction.

TorchRay. To encourage further research on the attribution problem and in partnership with Facebook AI Research, we open-sourced TorchRay[17] — a PyTorch package with various attribution methods and benchmarks implemented to support reproducible research.

1.2.4 Theme 2: Understanding the internal

The second theme is concerned with understanding the internal representation of the model.

Concept encoding. One of the ongoing debates within both the neuroscience community and the AI community is how relevant information is encoded in each community's learning system of interest — the human brain and a CNN respectively. One hypothesis is that a specific, relevant semantic concept (*e.g.* the face of my grandmother) is encoded by a single (or a few) neurons in my brain.[18] Another hypothesis is that concepts are encoded by *populations* of neurons (*i.e.* groups of neurons), not individual neurons. More formally, these two hypotheses represent two extremes on a spectrum, ranging from a sparse encoding (*i.e.* very few neurons do the work) to a distributed encoding (*i.e.* many neurons do the work), and beg the question, "How many neurons are required to encode a single concept?"

Neuron packing. Both research communities are also interested in the inverse question, "How many concepts does a single neuron encode?" This is informally known as the "neuron packing" problem,[19] as it asks how many concepts can be "packed" into a single neuron. Answering this question would elucidate whether neurons tend to be highly specific or multi-purpose.

[15] See this educational article for more on optogenetics.

[16] A 1 in a mask "expresses" or allows a specific neuron in a CNN to "fire" as it originally had; a 0 "blocks" that neuron's "expression."

[17] https://github.com/facebookresearch/TorchRay

[18] This is formally known as the "grandmother cell" theory.

[19] I first heard Chris Olah use this phase: see Chris Olah's research note on "Poly-Semantic Neurons."

An analog to medical imaging. One way to try to answer these two questions is to observe how neurons respond to specific concepts. This is analogous to how medical professionals and researchers use *molecular imaging* to "see" how an injected, molecular[20] stimuli moves through a patient's body via medical imaging (*e.g.* ultrasound, MRI). This kind of imaging allows medical professionals and researchers to observe where different molecules end up in the human body.

Similarly, one way to understand how concepts are encoded is to "see" which CNN neurons fire consistently when presented with examples of a specific concept (*e.g.* images of golden retrievers). Unlike medical researchers, who have limited access into a living human body, AI researchers typically have full access to a CNN, their model of interest.[21] One way to do this would be to learn how to combine neuron firings (*i.e.* activations) to perform a concept-specific task, such as *classifying* whether an image contains a golden retriever or *segmenting* the image regions in which a golden retriever appears.

1.2.4.1 Concept vectors

In chapter 5 (Fong et al., 2018b), we present **Net2Vec**, a novel method for learning **concept vectors** that successfully combines activations — by weighting and summing up individual neuron activations — to perform concept classification and segmentation. By design, our learned concept vectors assign a numerical weight to individual neurons that denotes its influence in encoding a particular concept (*e.g.* golden retriever). We gain several insights from learning these concept vectors.

First, we quantify *how many* neurons are needed to encode a specific concept using concept vectors. By varying the number of neurons used when learning a concept vector, we can observe when performance on the concept task (*e.g.* classification or segmentation) saturates. We find that the number of neurons at which performance saturates[22] varies by concept (*e.g.* 8 neurons for the "person" concept vs. 64 for "airplane"[23]).[24]

Second, using our segmentation concept vectors, we produce segmentation masks that highlight the presence of a concept (*e.g.* "dog"). We show that our visualization based on combining activity from neuron populations performs better

[20]Molecules are the smallest units that make up organisms or materials; thus a molecular stimulus is very small and on the scale of cells.

[21]The difficulty for AI researchers is not access, but rather interpretability (*i.e.* how do we make sense of the millions of parameters (*e.g.* neurons) we have access to?).

[22]Performance saturates when using more neurons does not improve performance.

[23]Number of neurons is out of 256 neurons in a late layer of an image classifier.

[24]This is similar to work done in Agrawal et al., 2014.

Figure 1.4: Comparison of segmenting with individual neurons (Bau et al., 2017) vs. combinations of neurons (Fong et al., 2018b). Here, we show an example of segmenting a boxer image (right) with the single, best "dog" neuron (left) and with a combination of neurons using the "dog" concept vector (middle, ours).

(*i.e.* higher-quality segmentation, see Figure 1.4) and is more *robust* than that based on activity from a single neuron.[25]

Third, we show how concept vectors allow us to *compare* how concepts are encoded both *within* a model and *between* different models. By performing arithmetic with concept vectors, we show that CNNs possess a coherent understanding of concepts and how they interact: for instance, a CNN's approximate solution of "sky" to the following arithmetic operations is coherent: "grass" + "blue" − "green" ≈ "sky." Furthermore, we demonstrate CNNs that learned in similar manners (*e.g.* taught to perform the same task) also understood concepts in more similar ways compared to those that learned in a different manner.[26,27]

1.2.5 Theme 3: Using exploratory tools

The third theme is concerned with developing visualization tools to enable AI practitioners and researchers to explore and understand the behavior of AI models quickly and easily.

An analog to medical equipment. This is analogous to the suite of medical devices and equipment that healthcare professionals rely on on a daily basis to help them triage medical issues. Consider the *stethoscope*, which enables a medical

[25]We show that Bau et al., 2017's single neuron visualization typically only produces salient segmentation masks for images that trigger extremely strong response in a single neuron.

[26]We quantified concept understanding by measuring the distance between concept vectors and storing these distances in a similarity metric. We then compared similarity matrices to quantify how similar the concept understandings of different models were.

[27]We found that the type of supervision (*e.g.* fully supervised vs. self-supervised) was one of the most influential differentiators in a model's understanding of concepts.

worker to listen to the internal sounds of the human body. A doctor can easily use a stethoscope on different locations in search for any abnormal signs. She may also use other tools (*e.g.* blood pressure monitor) in tandem with a stethoscope to provide more rich information about her patient. We desire a similar toolkit of visualization tools that allows for real-time inspection of a CNN and supports combining visualization techniques.

Interactive visualizations. In the past few decades, digital technology has quickly evolved, and the ways we interact with technology have also changed. For example, we have moved from using a keyboard to interact with a computer terminal to using a mouse to interact with a graphical user interface. Now, in the era of mobile devices and virtual assistants, we naturally use touchscreen gestures and voice commands (*e.g.* "Hey Alexa, ..."). However, as deep learning has rapidly advanced in the past decade, the ways we interact with and visualize deep learning models has remained static: most visualizations are non-responsive charts, images, and videos. Thus, we desire visualization tools that are as responsive and intuitive to use as today's touchscreen devices.

Representational similarity. With such a tool, one thing we might want to explore is how a model considers various inputs. For a CNN, this problem can be formulated as being able to quickly compare a model's representation (*i.e.* neuron activity) of different image regions. What is similar (and different) between CNN representations of two images that both contain golden retrievers?

1.2.5.1 Interactive similarity overlays

In chapter 6 (Fong et al., in prep), we present **interactive similarity overlays**, a simple, interactive technique that visualizes the representational similarity between different image patches. Leveraging the power of modern web technology (*e.g.* Javascript), given the current image patch that a mouse pointer is hovering over, we compute similarity scores with other image patches (*e.g.* in the same image and/or in other images) and visualize them using heatmaps overlaid on the original images (see fig. 1.5). Our method allows a user to drive their own exploration by hovering over various regions of interest; it can also be combined with other techniques. We demonstrate how to explore a few different phenomena.

First, by visualizing the same image at different locations in a CNN (*e.g.* network layers), we demonstrate how our technique can be used to explore and understand model representations at different depths.

Figure 1.5: Interactive similarity overlays. Our interactive visualization shows how similar (or different) a CNN considers different image patches to the current image patch (highlighted in yellow). The brightness of a patch denotes its similarity to the current patch (*i.e.* brighter denotes more similar, while darker denotes less similar). These images originally appear in Fong et al., in prep.

Second, by visualizing images containing the same object (*e.g.* a golden retriever), including generated images, and combining our method with another technique (*i.e.* matrix factorization) that highlights the main image components (*e.g.* the dog's head, body, and background), we demonstrate how to explore corresponding, intra-class[28] features across images.

Third, by visualizing the same image that has been systematically transformed (*e.g.* rotation, scale) and adding a simple line plot that visualizes the similarity scores of the same image patch upon transformation, we enable the discovery of features that are more or less sensitive to geometric transformations.

1.2.6 Focus on research

In the previous sections, we developed specific analogies between diagnostic tools used by medical professionals and the methods we introduce in this thesis; these analogies are chosen to help the lay reader understand at a high level the *function* of our techniques. However, the *purpose* of our methods is primarily *research*.

As noted in fig. 1.1, our role in the AI ecosystem is more analogous to that of medical researchers and scientists than that of medical professionals. Similar to how scientists study particular models, learning systems, or pathologies (*e.g.* a mouse model, the human visual system, cancer), we aim to add to scientific knowledge by developing a thorough understanding of a particular kind of AI system (*i.e.* CNNs). Just as some medical tools are used both by medical researchers and professionals (*e.g.* medical imaging), the methods we introduce can similarly be used by both AI researchers and practitioners. That said, our primary focus is research, and we aim to aid AI practitioners as a secondary focus.

[28] *i.e.* features within the same object class.

1.2.7 Structure of thesis

Our thesis is structured as follows: in chapter 2, we provide a thorough review of work related to ours. In the subsequent chapters (chapters 3 to 6), we present our original work on meaningful perturbations, extremal perturbations, concept vectors, and interactive similarity overlays. Finally, we discuss the impact of our work and promising future directions in chapter 7.

We also include a number of appendices. In appendix A, we provide a primer on relevant math concepts and notation, and in appendix B, we provide a primer on CNNs. These first two appendices are meant to be accessible notes to enable a reader with minimal background knowledge to follow at a high-level the mathematical details included in this thesis. That said, most of this thesis (*i.e.* all chapters except chapters 3 to 6) was written with minimal mathematical notation to encourage accessibility. In appendix C, we include a few other papers written during the course of this PhD that are related to the topics of improving CNNs and CNN interpretability respectively.

Lastly, for readers with little exposure to machine learning, we include an index of important terms and where they are defined in this thesis.

2
Literature Review

Contents

In this chapter, we review developments in *interpretability*[1] research on explaining and exploring a CNN model's behavior using 1. *"black-box"* visualizations (section 2.2); 2. *"white-box"* visualizations (section 2.3); and 3. *interactive* visualizations (section 2.4).

These thematic categories broadly correspond with the research themes of the works presented in this thesis (chapters 3 to 6) and introduced in section 1.2. To fully follow the details in sections 2.2 to 2.3, we recommend lay readers first

[1]The research field of *interpretability*, as it pertains to deep learning, refers to the work concerned with providing explanations (*i.e.* interpretations) of the behavior and inner workings of deep neural networks. See Gilpin et al., 2018 for a broad survey.

read appendices A to B. Given the scope of this thesis, we focus the bulk of our literature review on understanding *convolutional neural networks* (CNNs) for *visual data*.

Before diving into detailed reviews of these themes, we first provide a broad overview on work on explaining AI systems (section 2.1).

2.1 Explainable AI (XAI)

Explainable AI (XAI) research focuses on *explaining* AI systems.[2] Recently, as deep learning has become the dominant paradigm in AI, explainable AI has become more important. This is because the success of deep learning comes from its use of a large number of neural network parameters (*i.e.* weights), which enable a network to learn to model well a large amount of data and all the variety in it.[3] However, the highly-parameterized nature of deep neural networks also makes them difficult to understand. In comparison, some earlier machine learning models such as decision trees are more naturally *interpretable* (*i.e.* easy to understand) because they are both simpler in nature and intentionally designed to be interpretable.[4] This begs the question, can we have both performance and interpretability?

Broadly, there are three motivations for desiring explainable AI:

1. a scientific desire to *understand* an AI system;
2. a user-oriented desire to develop *trust* in an AI system; and
3. an educational desire to *learn* how to do something from an AI system.

Although these motivations can overlap, most interpretability research can be primarily categorized under one of these three reasons.

Understanding. The first motivation is similar to that of basic scientists (*e.g.* neuroscientists) studying living organisms: they are motivated to build deep knowledge about a given species, ranging from its typical behavior to its internal biology. The focus of this thesis falls under this category, and we develop methods to both characterize the external behavior of CNNs as well as understand their

[2] For a high-level overview on XAI, see Gunning, 2017; Samek et al., 2019; Arrieta et al., 2019.

[3] This also partially explains the surge in popularity of the terms "big data" and "data-driven machine learning."

[4] XAI research is often called *interpretability* research, and the two phrases are used somewhat interchangeably; we use "interpretability" to refer to work motivated by a scientific desire to understand an AI system.

internal mechanisms. The rest of this chapter will discuss work that is similarly motivated, so we postpone discussing this direction for now. In this thesis, we use the term "interpretability" primarily to refer to works motivated by the desire to understand models.

Trust. The second motivation is rooted in the relationship between consumers and producers: producers typically aim to persuade a consumer to use a product; in that quest, they often need to gain the trust of the consumer that the product will behave as expected. This is also relevant to regulators who aim to discover whether and how to enforce that products are indeed trustworthy.

In order to develop trust in deep learning, a number of researchers have focused on developing *explanation-producing* AI systems (Hendricks et al., 2016; Z. Zhang et al., 2017; Huk Park et al., 2018). Unlike basic deep learning models, which simply output a prediction, these systems aim to produce an explanation alongside every prediction.

However, because the primary motivation of such work is to develop trust in an AI system, such systems may produce *interpretable* (*i.e.* easy to understand) explanations that engender trust but do not *faithfully* (*i.e.* accurately) explain the actual behavior of the model. For example, a seasoned doctor may provide a short and interpretable explanation for their advice on a suggested diagnosis or treatment that may or may not fully capture their reasoning. This is because the purpose of the explanation is primarily to develop trust with their patient in order to persuade them that their suggestion is a wise course of action.

This apparent tension between *interpretability* and *faithfulness* of an explanation exists not only for trust-oriented XAI work but also the other two kinds of motivations for XAI.[5] Nevertheless, trust-oriented XAI research needs to be particularly cognizant of producing faithful explainations, as, depending on the actor, trust can be often in tension with faithful clarity.[6]

A number of explanation-producing systems (Brendel et al., 2019; Marcos et al., 2019) have been thoughtfully and intentionally designed such that the explanation produced is indeed a faithful explanation of the model's behavior; these models are often known as "inherently interpretable" or "interpretable-by-design." Such systems tend to be motivated by the desire to present a model whose decision-making process can be easily understood. Furthermore, a number of techniques

[5] A few works within the understanding-oriented tradition (Kindermans et al., 2019; Adebayo et al., 2018) have highlighted shortcomings of interpretability techniques that do not faithfully explain model behavior; we discuss this further in section 7.1.3.

[6] Consider the common tension between business and consumer interests.

originally developed to understand CNN behavior have also be used as explanations to engender trust (Zhou et al., 2018b; Kim et al., 2018).

See Brundage et al., 2020 for an extensive, accessible discussion on engendering trust in the AI development process using tools to verify claims about the behavior of AI systems.

Learning. A third motivation for XAI is a desire to learn from AI systems. As AI systems increasingly become more powerful, they will likely surpass human performance on certain tasks. In such scenarios, we may desire humans to learn skills from AI systems and thus require clear explanations from an AI system on how to perform a given task. For example, since AI systems have surpassed human abilities in playing chess and Go,[7] humans players have begun to study games played by AI systems to learn how to improve their own gameplay (Sadler et al., 2019).

Formally, *knowledge transfer* refers to transferring knowledge about something (*i.e.* how to perform a specific skill) from one entity to another (*e.g.* humans teaching other humans, AI systems teaching AI systems, humans teaching AI systems, AI systems teaching humans). Thus far, because AI systems — particularly robotic systems — have lagged behind humans in an number of tasks, most work has focused on transferring knowledge from humans to AI systems — *e.g.* teaching robots how to manipulate objects from human demonstration) (Skubic et al., 2000; Lee, 2017) — as well as transferring knowledge from strong AI systems to weaker ones — *e.g.* to *compress* the knowledge of a powerful AI model into one that can be run on a mobile device (Hinton et al., 2015; Cheng et al., 2017).

Another related topic is *human-machine collaboration*, in which humans and machine work together to accomplish various tasks that are difficult to accomplish on their own. Several XAI works fall under the category of developing "human-in-the-loop" AI systems, in which humans collaborate with an AI system (Kim, 2015; Lage et al., 2018), or "human-aligned" AI systems, in which models perform in a way more aligned to how humans make decisions (Scheirer et al., 2014; Fong et al., 2018a; Ross et al., 2017; Lage et al., 2018; Selvaraju et al., 2019).

Although there is relatively less emphasis on teaching humans to learn from AI systems in comparison to other XAI topics, some work has been done in this space, primarily in the form of *intelligent tutoring systems*, some of which use AI

[7]Similar to chess, Go is a two-player, abstract strategy board game that was invented in China over 2,000 years ago. See the Wikipedia article on "Go (game)."

to provide individual instruction to learners (Self, 1988; Hämäläinen et al., 2006).[8]

In conclusion, there are a variety of motivations for desiring explanations of and from AI models. In this thesis, we focus on explanations aimed at understanding the external behavior and inner workings of CNNs, one of the most popular state-of-the-art AI models today.

2.2 "Black-box" visualizations explaining external CNN behavior

"Black-box" explanations aim to characterize a model's behavior without accessing its inner workings (*i.e.* by treating it like a black box). They do this by reasoning the relationship between a model's inputs and outputs.

An *instance explanation* (*a.k.a. local explanation*[9]) is an explanation of a model's behavior on a particular example and is arguably the most well-known, black-box visualization technique.[10] For visual data, such an explanation most commonly takes the form of a *heatmap* that highlights what image regions are a responsible for a given prediction (see fig. 2.1).

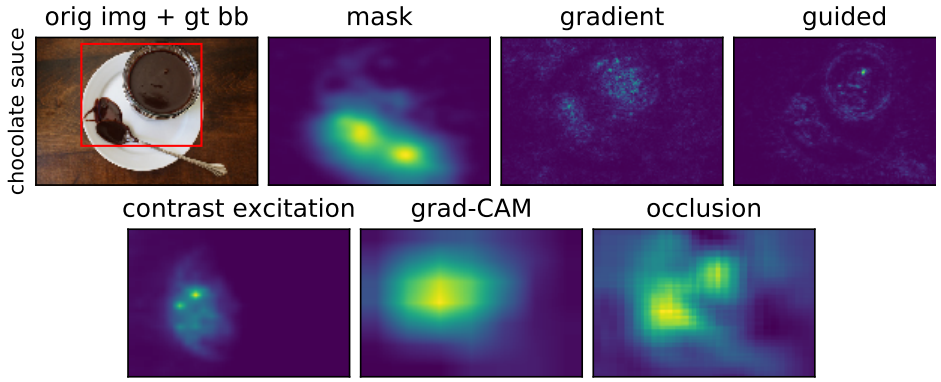


Figure 2.1: Examples of attribution heatmaps. Here, we show several saliency maps generated by different attribution methods to explain a CNN's "chocolate sauce" prediction (top row, 1st column: original image, with the ground truth bounding box in red). Top row: Fong et al., 2017; Simonyan et al., 2014; Springenberg et al., 2015; bottom row: J. Zhang et al., 2018; Selvaraju et al., 2017; Zeiler et al., 2014. Images originally appeared in Fong et al., 2017.

[8] See the Science Direct article on "Intelligent Tutoring System" and the Wikipedia article on "Intelligent tutoring system."

[9] A *local explanation* refers to one that explains how a model locally behaves around a specific input example.

[10] Other kinds of black-box visualizations typically explore the *global* relationship between model inputs and outputs; we discuss examples of such work in section 2.4.

Most **attribution heatmaps** — *a.k.a. saliency maps* — are generated in one of the following ways: 1. by visualizing a signal that has been *propagated* through the network; 2. by *perturbing* the input and observing the resultant effect on the model's output; and 3. by *approximating* the model's decision for a given example using an arguably more interpretable model.

Propagation				Perturbation			Approximation	
Visualizing gradient	**Relevance propagation**	**Averaging / integrating**	**Visualizing activations**	**Random occlusions**	**Iterative occlusions**	**Optimized occlusions**	**Propagation**	**Optimization**
Gradient (Sim14)	LRP (Bach15)	SmoothGrad (Smi17)	Linear Approx (Kin16)	Occlusion (Zei14)	Min Img (Zhou15)	Feedback Layers (Cao15)	Gradient (Sim14)	LIME (Rib16)
DeConvNet (Zei14)	Excitation BP (Zha16)	Integrated Grad (Sun17)	CAM (Zhou16)	RISE (Pet18)	*XRAI (Kap19)*	**Meaningful Pert (Fong17)**	Linear Approx (Kin16)	
Guided BP (Spr14)	DeepLIFT (Shr17)	Expected Grad (Stu20)	Grad-CAM (Sel17)			*Real-time Sal (Dab17)*		
Feedback Layers (Cao15)						**Extremal Pert (Fong19)**		

Figure 2.2: Overview of attribution methods. Saliency map techniques are organized by methodology (see section 2.2). **Bolded** work is ours (see chapters 3 to 4); *italicized* work is discussed in section 7.1.

2.2.1 Propagation-based methods

Propagation-based saliency techniques are the earliest and arguably most popular class of attribution methods for CNNs. This is because they are fast to compute, as they usually require only a single forward and backward pass through a network. These methods typically visualize a backpropagated signal (*i.e.* a gradient signal) and/or a forward-propagated signal (*i.e.* an activation tensor).[11]

2.2.1.1 Visualizing the gradient

The first of these methods, Simonyan et al., 2014's gradient visualization, focuses on visualizing the gradient of a network's output with respect to the input:[12,13]

$$\frac{\partial \Phi_c(\boldsymbol{x})}{\partial \boldsymbol{x}}. \tag{2.1}$$

Intuitively, the gradient visualizes how *sensitive* the output score, $\Phi_c(\boldsymbol{x})$, is to a small change in each pixel of the input image.[14] Visually, gradient visualizations

[11] Forward propagation of an input (*e.g.* an image) to a particular layer of a network yields a 3D activation tensor. Backward propagation (*a.k.a.* backpropagation) of an output to a network layer yields a 3D gradient tensor.

[12] Here, $\Phi_c : \mathbf{R}^{3\times H\times W} \to \mathbb{R}$ is the prediction of CNN Φ for class c and $\boldsymbol{x} \in \mathbf{R}^{3\times H\times W}$ is a RGB image. See appendix A.7 for more details about the notation used in this thesis.

[13] In practice, Simonyan et al., 2014's gradient method visualizes the magnitude of the maximum color channel k: Grad $::= \max_k |\frac{\partial \Phi_c(\boldsymbol{x})}{\partial \boldsymbol{x}}|$.

[14] This is also known as a *local* explanation, as it explains the behavior of a model for a small, local neighborhood around the given input $\boldsymbol{x}$.

are often quite noisy (see "gradient" example in fig. 2.1; best viewed online with zoom); thus, several subsequent works aim to reduce the noisy appearance and improve the visual quality of heatmaps.

Several early works, namely **DeConvNet** (Zeiler et al., 2014) and **Guided Backprop** (Springenberg et al., 2015)[15] do this by emphasizing the spatial locations that would *positively* improve the network's prediction (compare "guided" vs. "gradient" in fig. 2.1). In practice, this involves modifying the backpropagation rule for ReLU layers[16] such that the backpropagated gradient is only preserved for such spatial locations (see fig. 2.3).

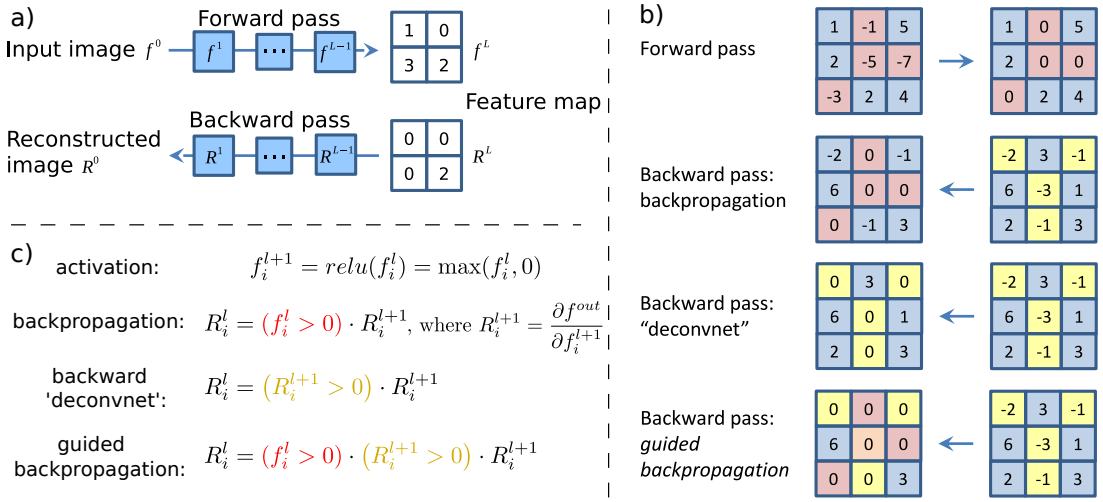


Figure 2.3: Overview of Gradient (Simonyan et al., 2014), DeConvNet (Zeiler et al., 2014), and Guided Backprop (Springenberg et al., 2015). To improve the quality of the gradient (*a.k.a.* backpropagation) visualization, deconvnet and guided backprop modify the backpropagation rule for ReLU layers. **(a)** A backprop-based heatmap R^0 is constructed from information in a backward pass through a CNN. **(b)** Examples of how several methods apply different backward rules for ReLUs. **(c)** Definitions of backward rules for ReLU layers used by different methods. f_i^l refers to the activation at location i after layer l; R_i^l refers to the backpropated signal at location i after layer l. Images reproduced with permission of Springenberg et al., 2015.

2.2.1.2 Mitigating gradient saturation

Another research stream seeks to improve the visual quality of attribution heatmaps by addressing the problem of *gradient saturation*. Whereas the gradient measures the *local sensitivity* of a feature (*i.e.* the sensitivity of the output with respect to a small

[15] Mahendran et al., 2016a's **SaliNet** method is equivalent to Springenberg et al., 2015's guided backprop.

[16] See appendix B.2.2 for more on the ReLU layer.

change in a feature), such as a pixel or spatial region in the input image, there are times when a feature may be globally important, yet its gradient is locally saturated. Consider a fully trained network, which typically makes highly confident predictions (*e.g.* 90% softmax probabilities are not uncommon). In this case, the gradient step to improve highly confident predictions will likely be small (see fig. 2.4a).

A number of techniques — *e.g.* **Layer-wise Relevance Propagation (LRP)** (Bach et al., 2015), **Excitation Backprop** (J. Zhang et al., 2018), and **DeepLIFT** (Shrikumar et al., 2017) – address this problem by backpropagating a *relevance signal* signal instead of a gradient. This relevance signal preserves a *conservation principle*; that is, the relevance signal should always sum to 1 at any point while it is being backpropagated. In order to do so, these methods must override the default rules used for backpropagating the gradient and introduce an extensive suite of new rules for every kind of layer in a given network. These methods differ from one another in their choice of how to compute the relevance signal.[17]

Another set of methods (Sundararajan et al., 2017; Smilkov et al., 2017b; Sturmfels et al., 2020) tackles the gradient saturation problem by *integrating* or *averaging* the gradient over a number of choices for the input image in order to better capture the global effect of a pixel's contribution.[18] In contrast to relevance propagation techniques, which require custom implementations of a number of backpropagation rules, these expectation-based techniques require no special backpropagation rules. However, because they integrate or average gradients of multiple samples and thus require multiple passes through a CNN, these methods are slower than most propagation-based techniques.

Integrated Gradients (Sundararajan et al., 2017) numerically integrates the gradients of a network applied to a number of images that linearly interpolate a baseline input $\boldsymbol{x}_0$ and the input being considered $\boldsymbol{x}$:[19]

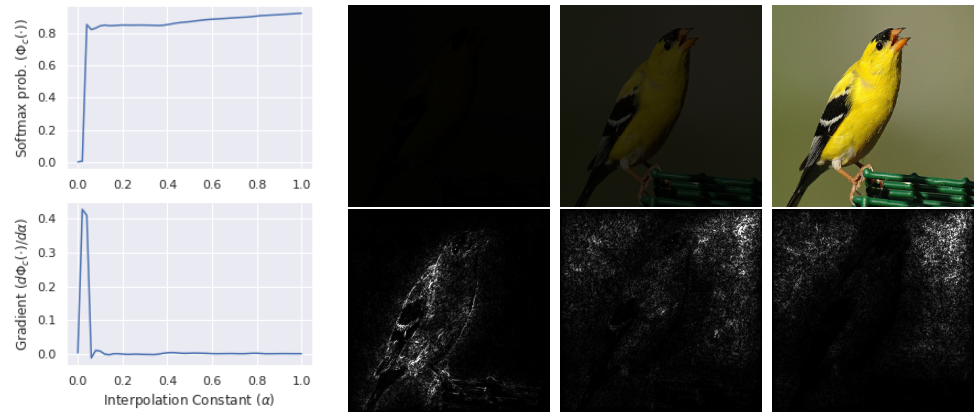
$$\text{IG} ::= (\boldsymbol{x} - \boldsymbol{x}_0) \odot \int_0^1 \frac{\partial \Phi_c(\boldsymbol{x}_0 + \alpha(\boldsymbol{x} - \boldsymbol{x}_0))}{\partial \boldsymbol{x}} d\alpha. \tag{2.2}$$

For images, a fully black image is used as the baseline. In practice, this means that the integrated gradients visualization for a given image accumulates the gradient

[17] That said, one of the variants of LRP, ϵ-LRP (Bach et al., 2015), is equivalent to excitation backprop (J. Zhang et al., 2018). The z-rule of LRP is also equivalent to linear approximation under certain conditions (*i.e.* network with ReLU activations and max-pooling layers) (Kindermans et al., 2016).

[18] These methods could also be categorized as perturbation-based methods; we class them as propagation-based methods because they more clearly evolved out of the propagation-based literature.

[19] $\odot$ denotes element-wise multiplication.



(a) Gradient saturation **(b)** $\alpha = 0.02$ **(c)** $\alpha = 0.20$ **(d)** $\alpha = 1.0$

Figure 2.4: Gradient saturation and interpolated images used in integrated gradients (Sundararajan et al., 2017). (a) The top chart plots a CNN's "goldfinch" prediction as a function the input image's intensity (controlled by α); this shows that the prediction saturates (*i.e.* plateaus) around $\alpha = 0.1$. The bottom chart shows that the gradient is small for $\alpha > 0.1$. **(b-d)** In the top row, we show images that are linearly interpolated between a baseline image (*i.e.* black image) and an image containing a goldfinch (rightmost column; see eq. (2.2)). In the bottom row, we show the gradient visualization for the "goldfinch" class for each interpolated image. We see that the gradient is most salient for $\alpha = 0.02$, which corresponds to where the gradient is large in (a). We also notice that the visualizations for $\alpha = 0.20$ and $\alpha = 1.0$ are very similar; this is because the gradient is small for $\alpha > 0.1$ (see (a)). Images reproduced with permission of Sturmfels et al., 2020.

when scaling the intensity of the original image, where α represents the intensity scaling factor (see fig. 2.4b-d).

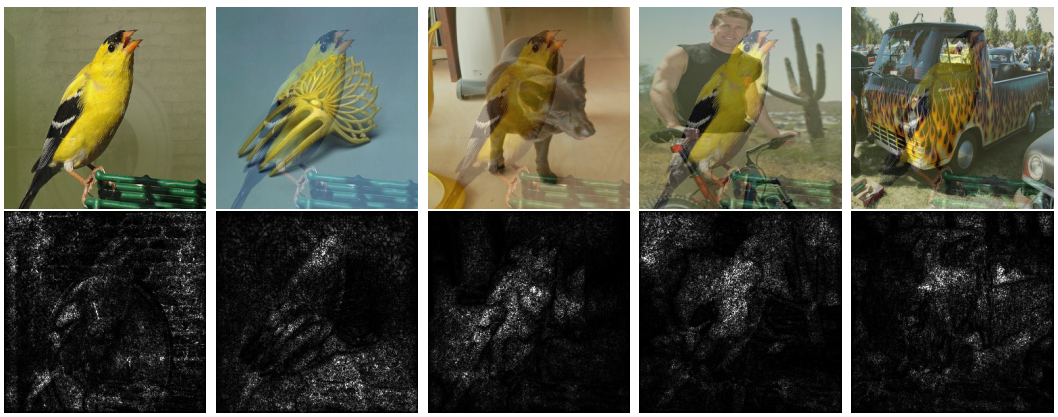
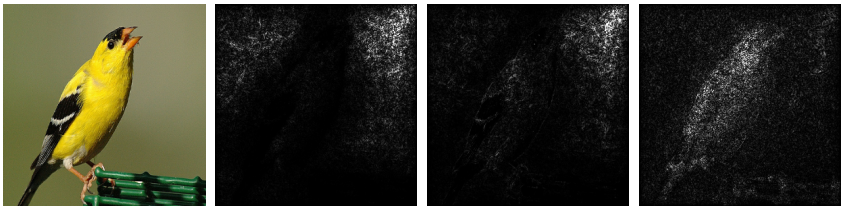


Figure 2.5: Interpolated images and gradients used for expected gradients (Sturmfels et al., 2020). Top: we show examples of the goldfinch image interpolated with random dataset images and with random α — *i.e.* $\boldsymbol{x}_0 + \alpha(\boldsymbol{x} - \boldsymbol{x}_0)$ in eq. (2.3)). Bottom: we show the corresponding gradient visualizations (Simonyan et al., 2014). Images reproduced with permission of Sturmfels et al., 2020.

Seeking to improve upon integrated gradients, Sturmfels et al., 2020 explore the space of possible baselines for images and show that integrated gradient heatmaps are dependent on the choice of the baseline image. To mitigate this effect, Sturmfels et al., 2020 introduce **Expected Gradients**. In contrast to integrated gradients, which interpolates between the black image and the image of interest, expected gradients accumulates gradients when applying a network to a number of uniformly random interpolations, $\alpha \sim \text{Uniform}(0, 1)$,[20] between a random image in the training dataset, $\boldsymbol{x}_0 \sim \mathcal{D}$, and the image of interest $\boldsymbol{x}$:

$$\text{EG} ::= \mathop{\mathbb{E}}_{\boldsymbol{x}_0 \sim \mathcal{D}, \alpha \sim \text{U}(0,1)} \left[(\boldsymbol{x} - \boldsymbol{x}_0) \odot \frac{\partial \Phi_c(\boldsymbol{x}_0 + \alpha(\boldsymbol{x} - \boldsymbol{x}_0))}{\partial \boldsymbol{x}} \right]. \tag{2.3}$$

See fig. 2.5 for examples of interpolated images and fig. 2.6 for a comparison with the gradient and integrated gradients methods.



(a) Image **(b)** Gradient **(c)** Integrated **(d)** Expected

Figure 2.6: Comparison of gradient (Simonyan et al., 2014), integrated gradients (Sundararajan et al., 2017), and expected gradients (Sturmfels et al., 2020). Here, we show the final visualizations of these methods. The visualizations for (c) and (d) were generated according to eq. (2.2) and eq. (2.3) respectively. Images reproduced with permission of Sturmfels et al., 2020.

Smilkov et al., 2017b introduces **SmoothGrad**, another expectation-based method that can be added to any existing attribution heatmap method. SmoothGrad seeks to improve visual quality of heatmaps (*i.e.* minimize noise) by adding noise. In particular, given an image, it creates copies of it and adds Gaussian noise to each copy. Then, it averages together the heatmap generated for each noisy copy, thereby "averaging away" the noise present in individual heatmaps (generated by $V_c(\cdot)$):

$$\text{SmoothGrad} ::= \frac{1}{N} \sum^{N} V_c(\boldsymbol{x} + \boldsymbol{z}), \text{ where } \boldsymbol{z} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2), \tag{2.4}$$

where $\boldsymbol{z}$[21] is noise tensor the same size as $\boldsymbol{x}$.

[20] See appendix A.5 for a primer on random numbers and distributions.

[21] See appendix A.5 for more on random variables that are independent and identically distributed (i.i.d.).

2.2.1.3 Visualizing with activations

Another class of propagation-based methods incorporates forward-propagated activations into their visualizations.

Linear Approximation (Kindermans et al., 2016) is arguably the simplest technique that uses activations:[22] it visualizes the "input × gradient" at a specific point in a CNN (*i.e.* at the input image or at an intermediate convoutional layer, where "input" is an activation tensor):

$$\text{LinearApprox} ::= \sum_k \Phi_k^l(\boldsymbol{x}) \odot \frac{\partial \Phi_c(\boldsymbol{x})}{\partial \Phi_k^l(\boldsymbol{x})}. \tag{2.5}$$

Whereas the gradient visualization highlights spatial locations that a classifier's prediction is most sensitive to (*i.e.* changes in those locations would significantly affect the prediction), the linear approximation highlights spatial locations that both contain salient features (*i.e.* strong activations) and decision-sensitive features (*i.e.* strong gradients).

Another method eschews gradients entirely and combines activations and fully-connected weights. Given a CNN that is comprised of *convolutional blocks* followed by a *Global Average Pooling (GAP)* layer and a single *fully-connected (FC)* layer, **Class Activation Maps (CAM)** (Zhou et al., 2016a) *linearly combines*[23] activations from the last convolutional layer, weighting them with their corresponding class-specific FC weights:

$$\text{CAM} ::= \sum_k w_k^c \Phi_k^l(\boldsymbol{x}), \tag{2.6}$$

where k denotes the index of a channel, $\Phi_k^l \in \mathbb{R}^{H_l \times W_l}$ denotes the spatial activation slice for channel k, and $w_k^c \in \mathbb{R}$ denotes the fully-connected weight connecting channel k to output class c. Intuitively, CAM highlights the spatial locations that contain class-relevant features (see fig. 2.7).

Grad-CAM (Selvaraju et al., 2017) extends CAM by generalizing the method to most CNN architectures, that is, those with *convolutional blocks* followed by *fully-connected* layer(s). Instead of using fully-connected weights to weight activations, Grad-CAM uses the gradient of the output class w.r.t. the activations at the last convolutional layer, after it has been globally averaged:

$$\text{Grad-CAM} ::= \text{ReLU}\left(\sum_k \alpha_k^c \cdot \Phi_k^l(\boldsymbol{x})\right), \text{ where } \alpha_k^c = \text{GAP}\left(\frac{\partial \Phi_c(\boldsymbol{x})}{\partial \Phi_k^l(\boldsymbol{x})}\right). \tag{2.7}$$

[22] The linear approximation visualization is closely related to the first-order Taylor expansion.
[23] See appendix A.4.1 for a primer on linear combinations.

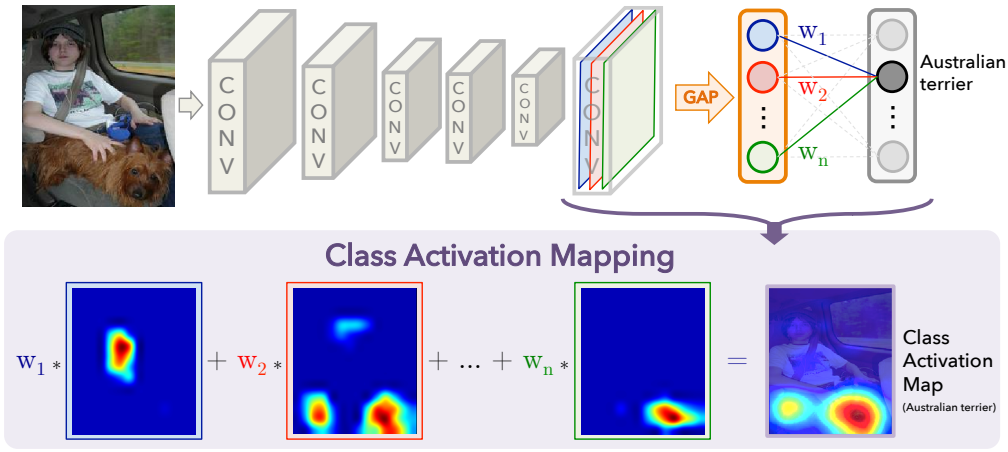


Figure 2.7: Overview of CAM (Zhou et al., 2016a). A CAM heatmap is generated by linearly combining each activation slice (*e.g.* heatmaps with colored borders) with the corresponding, class-specific fully-connected weight (*e.g.* $w_1, w_2, \ldots, w_n$). Images reproduced with permission of Zhou et al., 2016a.

Because CAM and Grad-CAM operate at a late CNN layer, where activations have low spatial resolution, their visualizations are typically resized[24] to match the size of the input image; this results in their "blobby" appearance (see fig. 2.7). Similar to DeConvNet (Zeiler et al., 2014), Grad-CAM highlights positive contributions to a model's prediction by thresholding the resultant heatmap with a ReLU function.

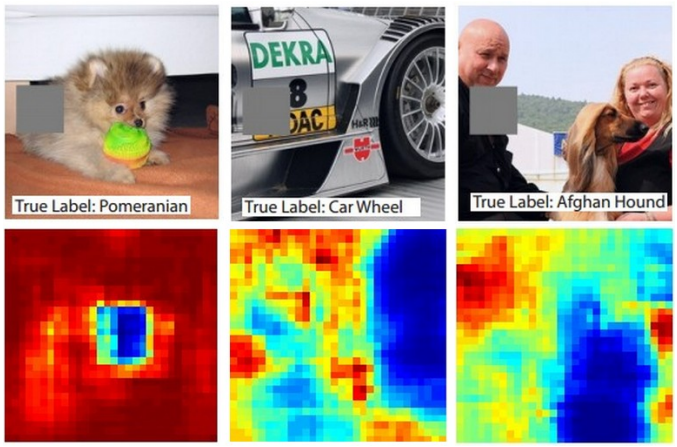
2.2.2 Perturbation-based methods

Another class of attribution methods is based on perturbations: Typically, an input image is perturbed numerous times; then, an attribution heatmap is synthesized by combining these perturbation regions with the corresponding changes to the output predictions. Unlike propagation-based techniques, perturbation-based methods tend to be more computationally expensive because they require multiple passes through a network (*e.g.* one pass is needed to observe the effects of one perturbation). Nevertheless, these techniques tend to have a *grounded meaning* because their heatmaps often have an explicit and clear interpretation; salient image regions denote that a model's output prediction is significantly affected when these regions are edited. They are often more model-agnostic, in that specific architectures or model modifications are not required.

Zeiler et al., 2014's **Occlusion** method was the first method of this kind. For this technique, a gray square is systematically moved across an image and the

[24] CAM and Grad-CAM use bilinear upsampling.

resulting class score is observed. A heatmap is generated by weighting the locations of where the gray square was placed with the corresponding class scores (see fig. 2.8).



.

Figure 2.8: Occlusion (Zeiler et al., 2014). To generate an occlusion visualization, a gray occluder box is moved over an image (top row) in a sliding window fashion and the changes in the CNN's predicted score for a particular class (*e.g.* "pomeranian") is observed and synthesized into a heatmap (bottom row). Here, the color of a pixel denotes the average output score when this pixel is occluded: blue denotes critical image regions that affect the prediction, as it corresponds with low confidence predictions (*e.g.* 20%), whereas red corresponds with high confidence (*e.g.* 90%). Images reproduced with permission of Zeiler et al., 2014.

One limitation of the occlusion technique is that only one specific kind of occlusion is considered: a single, fixed-sized square. In contrast, Zhou et al., 2015's **Minimal Image** method allows for image-specific, variable-sized perturbations to be considered (see fig. 2.9). Zhou et al., 2015 considers a minimal image to be a simplified version of an image of interest that captures the essential features needed (and nothing more) to be correctly classified by a model. To generate a minimal image, the following steps are taken: 1. image segments are automatically generated; 2. the segment that least impacts the model's prediction (*i.e.* least decreases the class score) is "deleted" from the image; and 3. step 2 is repeated (thereby removing unimportant image regions in an iterative fashion) until the model misclassifies the simplified image. In order to generate a smooth and natural-looking minimal image, Zhou et al., 2015 "deletes" image regions by editing the image in the gradient domain (Pérez et al., 2003) instead of the image domain.[25]

In part to consider a wider space of occlusions, Petsiuk et al., 2018 proposed Randomized Input Sampling for Explanation (**RISE**). Instead of masking with a

[25]This is also known as Poisson image editing. See the Wikipedia article on "Gradient-domain image processing" for more of an explanation.

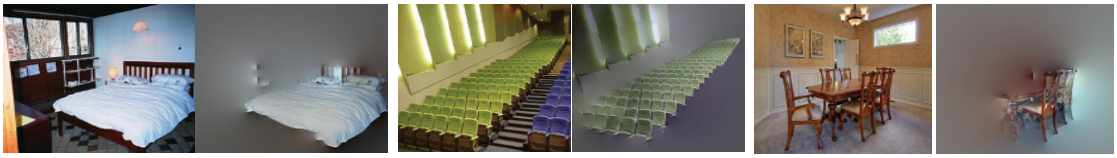


Figure 2.9: Minimal images (Zhou et al., 2015). Each side-by-side example shows the original image (left) and its minimal image (right); these were generated using a scene classifier with respect to the following categories: "bedroom," "art gallery," and "dining room." Images reproduced with permission of Zhou et al., 2015.

single square (Zeiler et al., 2014), RISE uses random binary masks in which spatial locations are randomly set to be "on" or "off" (*i.e.* 0 or 1, see fig. 2.10). Thus, RISE allows one to consider the effects of multiple, fixed-sized occlusions.

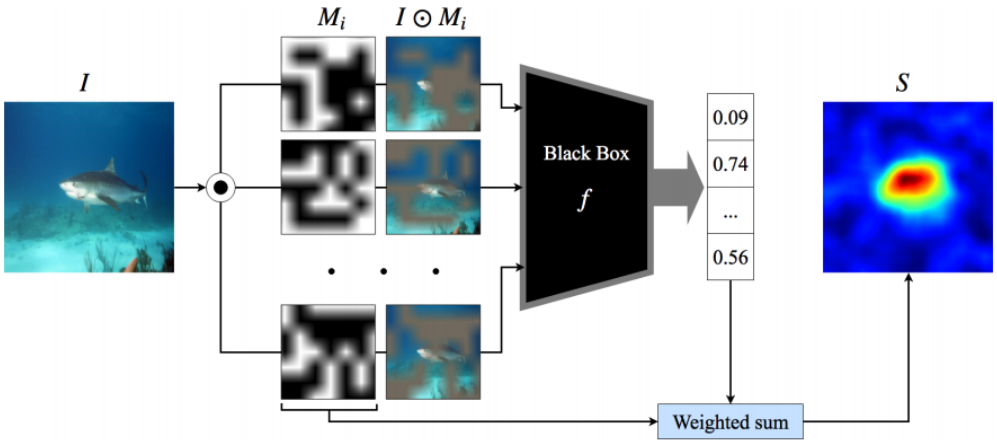


Figure 2.10: Overview of RISE (Petsiuk et al., 2018). The RISE visualization synthesizes a heatmap from observations of how a CNN prediction changes when image regions are randomly masked out from the input image (*i.e.* $I \odot M_i$). Images reproduced with permission of Petsiuk et al., 2018.

Rather that perturbing at the input (*i.e.* image) level, Cao et al., 2015's **Feedback Layers** perturb activations: they learn a binary masking layer (*a.k.a.* feedback layer) after every ReLU layer that element-wise masks activation tensors. These binary masks are optimized for a given image through multiple forward and backward passes through the network such that the target class score is maximized and that the binary masks are minimal (*i.e.* are mostly "off" or filled with 0s). Thus, feedback layers restrict the flow of information such that only class-relevant information is propagated. Finally, any propagation-based attribution heatmap can be generated through the feedback layers; Cao et al., 2015 visualize using the linear approximation technique.

2.2.2.1 Other applications of perturbations

Outside the realm of interpretability, some works leverage perturbations to improve weak and fully supervised localization.[26,27] While training an object classifier, **Adversarial Erasure** (Y. Wei et al., 2017) iteratively "deletes" parts of an image such that a given class score is minimized. It then uses the deleted image regions as a coarse training signal to train a segmentation network. Instead of masking at the input (*i.e.* image) level, **Fast-RCNN** (X. Wang et al., 2017) learns to occlude activations during training such that the class score is minimized. The network that generates occlusions adapts as an object detection network improves in performance; X. Wang et al., 2017 show that this adversarial[28] training paradigm improves object detection performance.

Others use perturbations to regularize a network, typically by randomly dropping out information in the input or at some intermediate part of the network.[29] **Hide-and-Seek** (Singh et al., 2017) and **Cutout** (DeVries et al., 2017) both "drop out" patches in the input image[30] in order to improve localization and classification performance respectively. Other methods occlude activation tensors at some point within a CNN. Srivastava et al., 2014 introduced **Dropout**, which is a popular technique for randomly dropping out individual voxels in an activation tensor and inspired a number of follow up works. **Spatial Dropout** (Tompson et al., 2015) drops out entire spatial slices associated to random filters. **DropPath** (Larsson et al., 2017) randomly drops a branch of information flow within a network; **Scheduled DropPath** (Zoph et al., 2018) extends this by linearly increasing the probability a branch is dropped throughout training. Leveraging the insight that spatial information in natural image is highly structured and locally smooth,

[26] *Localization* refers to the task of localizing where an object is in an image (as opposed to *classification*, which refers to the simpler task of just naming the dominant object in an image). Strictly, localization refers to predicting the segmentation mask of an object, while object detection refers to predicting a bounding box. However, it can also be used to describe both tasks.

[27] *Weak supervision* means that the ground truth labels provided are relatively "weaker" than the expected output. In the case of localization, weak supervision typically refers to image-level classification labels that name the objects; the locations of objects are not provided. In contrast, *full supervision* denotes that the labels used in training are the same as the predicted outputs (*e.g.* bounding boxes or segmentation masks for localization).

[28] *Adversarial training* refers to any training set-up such that one component aims to improve performance and another component aims to inhibit performance.

[29] *Regularization* refers to techniques to improve the robustness of a model (*i.e.* make a model work as expected on more diverse inputs). This is important so that a model does not *overfit* to the data used to train it but rather can generalize to novel examples.

[30] They differ in the way they drop out patches: Hide-and-Seek drops patches out stochastically from a fixed, super-imposed grid. Cutout drops out a fixed number of patches, randomly choosing each patch's position.

DropBlock (Ghiasi et al., 2018) drops out contiguous patches of voxels, as opposed to individual voxels as in dropout (Srivastava et al., 2014), and can be viewed as an extension of cutout (DeVries et al., 2017).

2.2.3 Approximation-based methods

Another class of attribution techniques approximates the local behavior of a model with an arguably simpler and more interpretable model. The gradient (Simonyan et al., 2014) and linear approximation (Kindermans et al., 2016) can be viewed as approximation methods, as they approximate the linear behavior around the input image. **LIME** (local intepretable model-agnostic explanations) (Ribeiro et al., 2016) learns an explicit linear model to approximate the behavior around a given input. For images, LIME segments an input image into superpixels and then learns a sparse linear model g that takes as input a vector of 1s and 0s, where each value corresponds to whether a given superpixel is shown or occluded, and is trained to approximate the behavior of a CNN. It is trained by sampling perturbations to the input (*i.e.* superpixels are randomly turned on and off), and thus can also be viewed as a perturbation-based attribution method. The weights of g, which correspond to superpixels, are then presented as the explanation. In theory, other models beyond linear ones (*e.g.* random forests) can be used to approximate CNN behavior; however, in practice, they often are less suitable for visual data.

2.3 "White-box" visualizations of internal CNN components

In contrast to "black-box" explanations, which typically characterize a model's external behavior by explaining the relationship between its inputs and outputs, *"white-box"* explanations leverage their "white-box" access to the inner workings of a model.

In this section, we survey works that focus on characterizing the internal components of a CNN. We highlight three main research directions: 1. visualization techniques that characterize the patterns that highly activate individual CNN filters and/or combinations of filters; 2. work on describing how semantic concepts are encoded in the internal representation of a CNN; and 3. research on understanding how sensitivity or invariant a CNN is to geometric tranformations.

2.3.1 Visualizing intermediate representations

Numerous works introduce visualization methods that highlight what a single CNN filter[31] "prefers" (*i.e.* is highly activated by). Many of these techniques have also been used to characterize what *directions* in activation space (*i.e.* combinations of filters) are selective for. Such works typically use either real examples or generated ones to describe what a CNN filter or filter direction is excited by.

We first formally define *filter directions* before preceding to review relevant works. Given an 3D activation tensor with k channels that is the output of an intermediate layer in a CNN $\Phi^l(\cdot)$, a *filter direction* can be defined as a 1D directional vector:

$$\boldsymbol{v} \in \mathbb{R}^k. \tag{2.8}$$

Then, when we say that an image $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$ is highly activated along a filter direction, we mean that the dot product between the filter direction and an activation tensor $\boldsymbol{a} = \Phi^l(\boldsymbol{x})$ — *i.e.* $\langle \boldsymbol{v}, \boldsymbol{a} \rangle$[32] — is large. (*i.e.* larger than the result of taking linear combinations with most other natural images). Intuitively, this denotes that the activation tensor points in a similar direction as the filter direction $\boldsymbol{v}$ A filter direction can be viewed as describing a specific way of combining individual filters.

To represent a single filter direction using this notation, the filter direction would be a one-hot vector (*a.k.a.* a natural basis vector) — *i.e.* filled with 0s except at a location k, where it is filled with a 1. Intuitively, such a vector denotes the direction that corresponds with filter k.

2.3.1.1 Visualization via real examples

Several works use real examples to describe the patterns that a filter or filter direction is highly activated by. Zeiler et al., 2014 show the image patches (*a.k.a.* **top activated patches**) that most activated a given filter as well as visualizing a DeConvNet attribution heatmap w.r.t. the given filter.[33] Zhou et al., 2015 builds on Zeiler et al., 2014 by learning the **empirical receptive field** for a given filter. They identify the top activated images for a given filter, extensively occlude them

[31] In this dissertation, "filter," "unit," "feature," and "channel" are used interchangeably to describe either the non-spatial dimension of an activation tensor or the set of corresponding convolutional weights that corresponds to that activation dimension. Precise usage would probably refer to the weights as either "filters" or "units" and to the associated output of those weights as a "channel" or "feature."

[32] If $\boldsymbol{a}$ has spatial dimensions (*i.e.* 3D tensor), then each scalar in $\boldsymbol{v}$ is broadcast to all spatial locations in each corresponding channel.

[33] Although attribution heatmaps are typically computed with respect to an output class unit, they can be computed with respect to any component of a model, including an intermediate filter.

and observing which regions most affect that filter's output activation — in a similar fashion to Zeiler et al., 2014; DeVries et al., 2017; Petsiuk et al., 2018 — and then summarize these effects in a heatmap. Using this technique, Zhou et al., 2015 also segments images based on their empirical receptive fields to highlight the regions that most activate a given unit. Bau et al., 2017 introduced **Network Dissection**, another technique that produces filter-specific segmentations of images. Using an auxiliary dataset with annotated semantic concepts, Bau et al., 2017 "probe" a network by quantifying the ability of thresholded, extremal filter activations (*i.e.* 99.5% quantile) to segment annotated concepts. These filter-based segmentations are then used to visualize how a unit is selective for a given concept (see fig. 2.11).

Figure 2.11: Examples from Network Dissection (Bau et al., 2017). We show real examples for which a single filter is highly activated and consistently corresponds with a semantic concept (*e.g.* top: "faces"; bottom: "staircase railings"). The segmentation masks are thresholded activations from conv5 filters 137 (top) and 52 (bottom) of a scene classifier. Such visualizations suggest individual filters correspond with specific, semantic concepts. Images reproduced with permission of Bau et al., 2017.

2.3.1.2 Visualization via generated examples

Most work on visualizing filters or filter directions *generate* examples that highly activate a filter or filter direction. These works collectively known as *feature visualizations*, as they typically visualize a CNN feature (*i.e.* filter) or direction in feature space.

Broadly, there are three main kinds of feature visualizations that differ based on what they aim to visualize: 1. activation maximization, which visualizes an example that maximally activates a filter or filter direction; 2. feature inversion, which visualizes the CNN representation (*i.e.* whole activation tensor) of an image at a given layer; and 3. caricatures, which augments an image with exaggerated features that highly activate the feature directions in the original image.

Furthermore, there are two main ways of generating feature visualizations: 1. via direct optimization, in which an image is directly learned (*i.e.* image pixels are optimized); and 2. via a generator network, which generates the feature visualization image and acts as a *strong natural image prior* (*i.e.* encourages the image to look realistic).

In this section, we first expand on the three basic kinds of feature visualizations and highlight direct optimization methods. Then, we proceed to discuss works that use generator networks to produce feature visualizations.

Direct optimization. The earliest feature visualization works *directly* optimize the pixels of a generated image.

Instead of showing real image patches that highly activate a given filter, Simonyan et al., 2014 generate an input image that most activate an output class unit. This technique is generally known as **activation maximization**. Intuitively, activation maximization characterizes the patterns that most activate a feature direction. This is somewhat analogous to scientific work that correlates the strong neural activity of a living organism (*i.e.* a monkey) with specific visual stimuli (*i.e.* images of bananas). Figure 2.12 shows examples that maximally activate a variety of CNN filters from different layers.

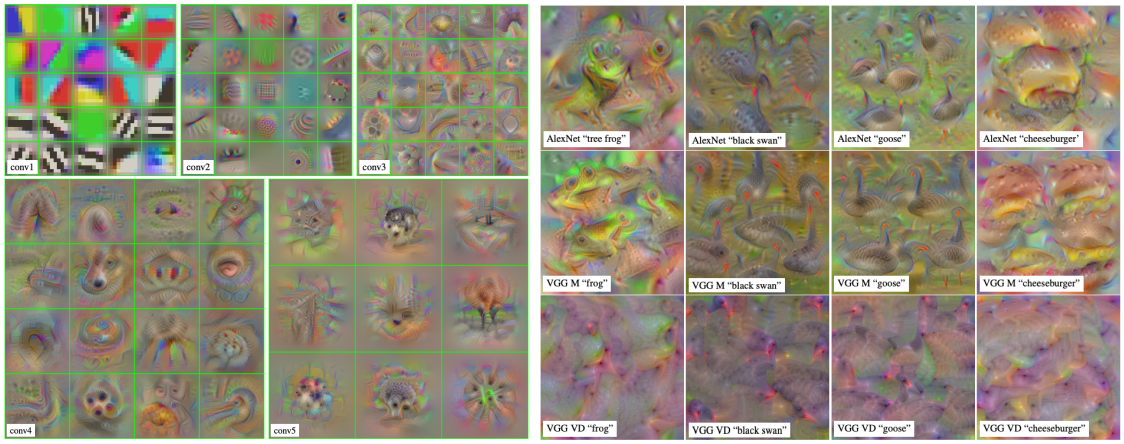


Figure 2.12: Activation maximization. Left: We show examples of activation maximization visualizations for several CNN layers. Notice how the visualizations become more complex as the layer depth increases. Right: We show examples of activation maximization with respect to a CNN's output class prediction. Images reproduced with permission of Mahendran et al., 2016b.

Specifically, Simonyan et al., 2014 learn an input image $\boldsymbol{x}^*$ that maximizes the output class score and had a small L_2 norm according to the following equation:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \Phi_c(\boldsymbol{x}) - \lambda \|\boldsymbol{x}\|_2^2. \tag{2.9}$$

The L_2 term acts as a regularizer[34] to prevent the input image from having large values and thus looking unnatural.

Beyond visualizing output class neurons, activation maximization can be applied to any intermediate filter or filter activation, by generalizing eq. (2.9). Given a filter direction $\boldsymbol{v}$, the image that maximally activates that direction is given by

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \langle \boldsymbol{v}, \Phi^l(\boldsymbol{x}) \rangle - \lambda \mathcal{R}(\boldsymbol{x}), \tag{2.10}$$

where $\mathcal{R}(\cdot)$ regularizes the generated image.

Mahendran et al., 2015 introduce the second kind of feature visualization: **feature inversion** (*a.k.a.* **representation inversion** or **representation reconstruction**). This technique generates an input image that matches an intermediate activation tensor of a reference image. Intuitively, a feature inversion shows a user how a CNN "sees" an image at a particular layer. Figure 2.13 shows feature inversions for an image at every layer in a CNN; these examples suggest that deep layers in a CNN do not remember the textural details of the input image; instead, they appear to remember semantic concepts (*i.e.* eyes, fur).

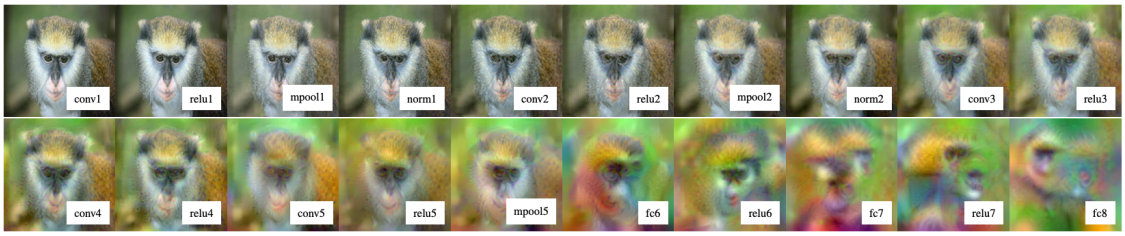


Figure 2.13: Feature inversion (Mahendran et al., 2016b). We show feature inversions of an image of a monkey at every layer of a CNN; notice how the visualizations become more abstract as layer depth increases. Images reproduced with permission of Mahendran et al., 2016b.

Formally, a feature inversion is learned by minimizing the following:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \mathcal{L}(\Phi^l(\boldsymbol{x}), \Phi^l(\boldsymbol{x}_0)) + \lambda \mathcal{R}(\boldsymbol{x}), \tag{2.11}$$

where the first term is a loss term that compares the internal representation at some intermediate layer l of a reference image $\boldsymbol{x}_0$ and the generated image $\boldsymbol{x}^*$ while the second term is a regularizer that encourages $\boldsymbol{x}$ to naturalistic. Mahendran et al., 2015 notably introduces a total variation regularization term, which encourages

[34]Here, a regularizer is an additional penalty term that encourages a certain property on a quantity (*i.e.* $\boldsymbol{x}$ should not have extreme values); this is also known as encoding a *prior*.

neighboring pixels in the learned image to be similar, thereby encouraging the learned image to be smooth.[35]

Mordvintsev et al., 2015 introduced a third kind of visualization known as **caricatures** (*a.k.a.* **deep dream**), which aims to exaggerate the features captured by a given network layer that exist in a reference image. Intuitively, these visualizations are somewhat analagous to caricatures created by artists, which exaggerate existing features in a subject. Figure 2.14 shows an example of a caricature visualization.

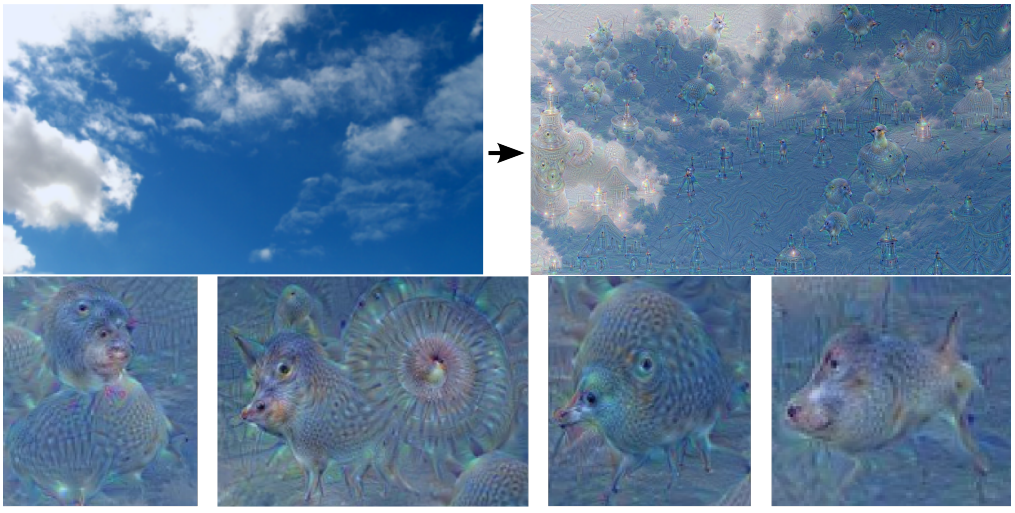


Figure 2.14: Caricature (Mordvintsev et al., 2015). We show a caricature (top right) of an image containing clouds (top left: original image). We zoom in and highlight a few interesting visualizations that pop up (bottom row). Much like caricatures drawn by visual artists, these feature caricatures (*a.k.a.* "deep dream") accentuate and exaggerate patterns found in the original image. Images reproduced from Mordvintsev et al., 2015 under CC BY 4.0.

Caricatures are learned in the same way as activation maximization visualizations with two key differences. First, instead of initializing the image being generated to one filled with random noise (as is typically done in activation maximization and feature inversion), the image being learned is initialized to the reference image itself. Second, instead of maximizing along specific filter direction $\boldsymbol{v}$, caricatures instead maximize activations in the direction of the reference image's CNN representation,

[35] Mahendran et al., 2015 uses a normalized Euclidean distance (*i.e.* $\mathcal{L}(\Phi^l(\boldsymbol{x}), \Phi^l(\boldsymbol{x}_0)) = \frac{\|\Phi^l(\boldsymbol{x}) - \Phi^l(\boldsymbol{x}_0)\|_2}{\|\Phi^l(\boldsymbol{x}_0)\|_2}$) as the loss term as well as an α-norm (*i.e.* $\mathcal{R}_\alpha(\boldsymbol{x}; \alpha) = \|\boldsymbol{x}\|_\alpha^\alpha$) and total variational (TV) norm (*i.e.* $\mathcal{R}_{TV}(\boldsymbol{x}; \beta) = \sum_{i,j}((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2)^{\frac{\beta}{2}}$), as regularizers, which encourage the learned image to have non-extreme and smooth values respectively.

$\Phi^l(\boldsymbol{x}_0)$. The resulting maximization is as follows:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \langle \Phi^l(\boldsymbol{x}_0), \Phi^l(\boldsymbol{x}) \rangle - \lambda \mathcal{R}(\boldsymbol{x}). \tag{2.12}$$

Together, these two key differences ensure that the resulting visualization appears very similar to the reference image (*i.e.* an exaggerated version of it).

While these feature visualizations are interesting (see figs. 2.12 to 2.14), they are often not visually realistic (*i.e.* they don't look like real images). Thus, since Simonyan et al., 2014; Mahendran et al., 2015; Mordvintsev et al., 2015, a number of subsequent works sought to improve the visual quality of these feature visualizations. They tend to fall into one of a few categories:[36] 1. frequency penalization; 2. transformation robustness; 3. strong natural image priors; and 4. diversity.

First, *frequency penalization* refers to mitigating the high-frequency artifacts that are typically present in feature visualizations (*i.e.* the highly pixelated, unnatural patterns). This can be done using regularization functions, like total variational norm (Mahendran et al., 2015; Nguyen et al., 2015; Øygard, 2015; Tyka, 2016; Mordvintsev, 2016; Olah et al., 2017).

Second, *transformation robustness* refers to learning an input image that generally characterizes the preferred stimuli of a filter, even if subject to geometric transformations (such as changes in color, scale, and jitter) (Mordvintsev et al., 2015; Olah et al., 2017). Then, the feature visualization can be seen as a representative exemplar of the kinds of input images a filter is selective for (*i.e.* it is selective for images that generally look like this, regardless of geometric transformations). This is typically encouraged by randomly transforming the generated image at every step of the optimization.

Third, *strong priors* refers to learning and utilizing a strong natural image prior (*e.g.* via a generator network) to encourage learned images to be naturalistic (Nguyen et al., 2016a; Nguyen et al., 2017; Ulyanov et al., 2018; Mordvintsev et al., 2018).

Lastly, *diversity* refers to generating a diverse set of feature visualizations, *e.g.* for activation maximization, in order to better characterize the diversity of stimuli that a filter is selective for (D. Wei et al., n.d.; Nguyen et al., 2016b; Nguyen et al., 2017; Olah et al., 2017). This is typically done by adding a diversity term to the optimization procedure.

[36] Olah et al., 2017 first used the first three groupings.

Optimization via generator networks. In the rest of this section, we focus primarily on research developments that use of generator networks as strong natural image priors in order to learn more naturalistic images.[37]

Instead of directly learning an image that reconstructs CNN features (as done in Mahendran et al., 2015), Dosovitskiy et al., 2016a trains a generator network to do feature reconstruction. Given a real image $\boldsymbol{x}_0$ and a CNN Φ being explained, the generator network takes as input the activation tensor of the image at a specific layer (*i.e.* $\Phi^l(\boldsymbol{x}_0)$) and outputs a generated image that is encouraged to be similar to the real one.[38,39]

Building on Simonyan et al., 2014 and Dosovitskiy et al., 2016a, Nguyen et al., 2016a improves upon activation maximization by using the generator network from Dosovitskiy et al., 2016a to generate an image that maximally activates an output class unit. In contrast to Simonyan et al., 2014, which directly learned an input image to maximally activate an output class unit, Nguyen et al., 2016a's Deep Generator Network-based Activation Maximization (**DGN-AM**) instead learns an input code to the generator network that produces an image that then maximally activates an output class (see fig. 2.15):[40]

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z}} (\Phi_c(G(\boldsymbol{z})) - \lambda \|\boldsymbol{z}\|_2^2). \tag{2.13}$$

Because a pre-trained generator network is used, the produced images are more realistic compared to those that are directly learned (compare fig. 2.12 with fig. 2.16).

Similar to Nguyen et al., 2015, Ulyanov et al., 2018's **Deep Image Prior (DIP)** uses a generator network to generate an image that maximizes a class activation (*i.e.* activation maximization). It also demonstrates how to use a generator network to produce an image that inverts a reference's image's feature representation (*i.e.* feature inversion). However, instead of optimizing the input code to the generator, Ulyanov et al., 2018 optimize the parameters of an untrained generator.

[37] See Olah et al., 2017 for a more thorough treatment of the other research advancements.

[38] Specifically, the generated image is encouraged to be similar to the input image in image space, feature space, and adversarially. Image space similarity means that pixels between the generated and reference image should be similar (*e.g.* via a L2 loss term). Feature space similarity means that intermediate activation tensors yielded by the two images should be similar (*e.g.* via a loss like that of the first term in eq. (2.11)). Adversarial similarity means that the distribution of generated images and that of real images should be indistinguishable (*e.g.* via an adversarial loss (I. Goodfellow et al., 2014)).

[39] Dosovitskiy et al., 2016a builds off Dosovitskiy et al., 2016b, which learns a generator that generates images from activation tensors to be similar in image space only, by adding a feature space (*a.k.a.* perceptual) loss and an adversarial loss.

[40] Nguyen et al., 2017 improves upon Nguyen et al., 2016a by using autoencoders as the generator class and generating images conditional on a class (as opposed to maximizing the class probability). In practice, this yields much more diverse images.

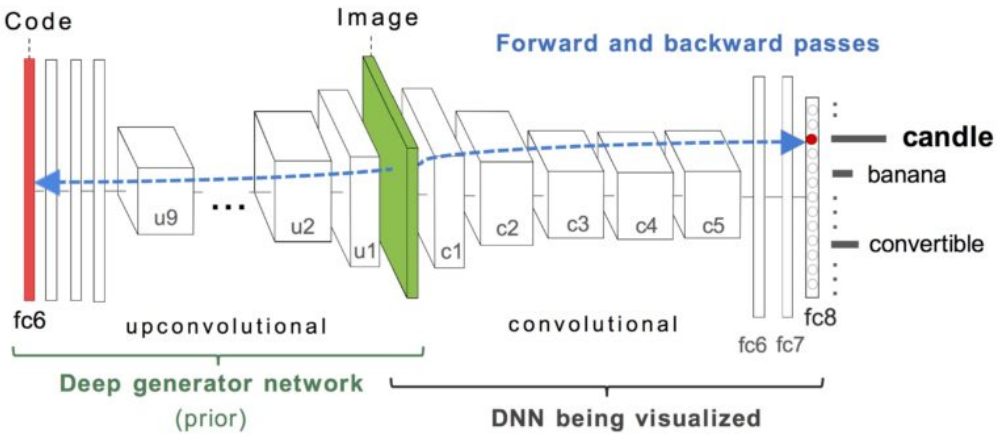


Figure 2.15: DGN-AM (Nguyen et al., 2016a). Instead of directly optimizing an image, as done in Zeiler et al., 2014, DGN-AM optimizes an input code that is used to generate an image such that the generated image maximally activates a CNN feature (*e.g.* the "candle" output class). The pre-trained generator network acts as a strong natural image prior, since it was trained to generate realistic images (see fig. 2.16 for generated examples). Images reproduced with permission of Nguyen et al., 2016a.

Figure 2.16: Activation maximization using strong natural image prior (Nguyen et al., 2016a). DGN-AM (Nguyen et al., 2016a) generates images that maximally activate specific CNN filters. Here, we show examples that highly activate conv5 filters (top) and scene output neurons (bottom). Their realistic appearance (as compared to examples that were directly optimized in fig. 2.12) is due to the use of a pre-trained generator network that acts as a strong natural image prior. Images reproduced with permission of Nguyen et al., 2016a.

For activation maximization, the following optimization is performed:

$$\theta^* = \arg\max_{\theta} \Phi_c(G(\boldsymbol{z};\theta)), \tag{2.14}$$

where θ^* denotes generator G's parameters and $\boldsymbol{z}$ is fixed as a randomly initialized

vector.[41] For feature inversion, the optimization is updated as follows:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\Phi^l(G(\boldsymbol{z};\theta)), \Phi^l(\boldsymbol{x}_0)) \tag{2.15}$$

where $\mathcal{L}$ is an L2 reconstruction error.

Using the same formulation in which the parameters of an untrained generator are learned, Mordvintsev et al., 2018 apply DIP to Compositional Pattern Producing Networks (CPPNs) (Stanley, 2007) to generate arbitrarily large images for activation maximization.[42]

Using a similar paradigm to Network Dissection (Bau et al., 2017), **GAN Dissection** (Bau et al., 2019b) identifies hidden units in the generator netwok of a GAN that are highly correlated with semantic concepts (*e.g.* trees, domes) and increases or decreases their activations at specific spatial locations in order to control the appearance of the generated image (*e.g.* to delete trees or add domes).

2.3.2 Concept encoding

A number of works sought to understand how semantic concepts are encoded internally in a CNN representation. Within neuroscience, the "grandmother cell" (Konorski, 1967; Gross, 2002) theory suggests that there are neurons fire when a specific, highly semantic visual concept is seen (*i.e.* the face of one's grandma or a specific person, like Jennifer Aniston). This theory represents the extreme end of the idea of *sparse* representations; the opposing view is that a specific concept is encoded by a *distributed* pattern across a large population of neurons.

In part because of the long line of activation maximization work, which typically visualizes what a single filter is selective for, several works argue that semantic concepts are encoded in a sparse manner by individual hidden units. Zhou et al., 2015; Zhou et al., 2018a leverage their visualizations which overlay segmentation masks over highly activated patches to support this argument.

Zhou et al., 2015 show annotators a set of images with their segmentation overlays highlighting where they are highly activated for a given filter in order to name and categorize the semantic concept that filter is selective for as well as to quantify how selective it is for that concept, by prompting annotators to mark images that don't fit. From this annotation experiment, they show that around 60% of units in each layer yielded high precision ($> 75\%$); they also show that filters

[41] That is, $\boldsymbol{z} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$.

[42] Unlike most generator networks, which typically produce a fixed size output, CPPNs can generate arbitrarily large images. It does this by taking as input an (x, y) coordinate and outputting the (r, g, b) pixel value for that coordinate.

from early layers were annotated to encode simpler concepts (*i.e.* colors) while later layers tended to encode more complex concepts (*i.e.* objects and scenes).

Using a dataset annotated with semantic concepts, Zhou et al., 2018a evaluate segmentation performance when segmenting concepts with thresholded, extremal filter activations in order to answer the question: "which filters are highly correlated with what concepts?" They found a significant portion of filters to be selective for unique concepts. Furthermore, they also show that, when activations are rotated randomly (*i.e.* by a random change-of basis), a number of rotated filters that are concept detectors significantly decreases; based on this experiment, they argue that trained CNNs learn filters that are highly semantic.

In contrast, Szegedy et al., 2014 visualize patches that are most aligned to a random direction in activation space and show qualitatively that random directions also corresponded with semantic concepts.

Agrawal et al., 2014 considered both encoding perspectives: First, they plot *precision-recall curves*[43] of individual filters when using their activations for an object detection task on the relatively small PASCAL dataset (Everingham et al., 2015); they found only a few "grandmother cell"-like units for which precision did not immediately drop as recall increased (see fig. 2.17a).[44] Second, they also trained a linear *support vector machine (SVM)*[45] on subsets of filters, from using a single filter to all filters and by ordering filters based on a heuristic of how important that filter is for detecting a given object.[46] Then, they plot object detection performance against the number of filters used and found that, for most classes, a substantial number of filters are needed to acheive near peak performance (see fig. 2.17b).[47]

Similar to Zhou et al., 2015, Gonzalez-Garcia et al., 2016 run a human annotation experiment to identify if extremal filter activations systematically correspond to a semantic object part (*e.g.* a wheel in a car). However, unlike Zhou et al., 2015, which

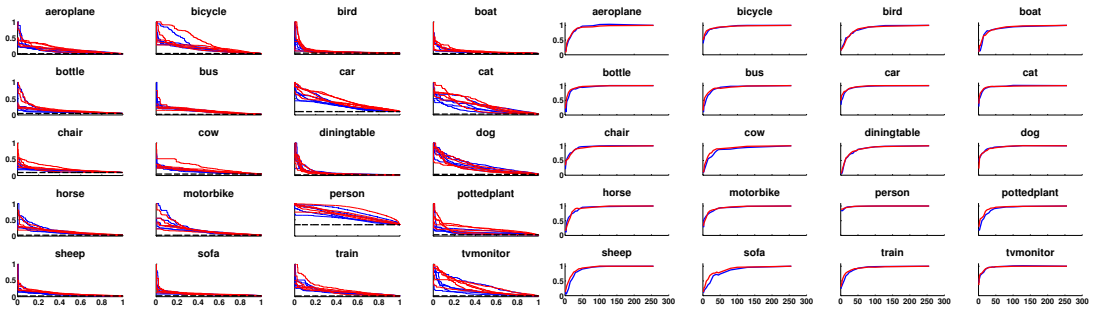
[43] *Precision* is the fraction of true positives out of all predicted positives (*i.e.* true positives + false positives), while *recall* is the fraction of true positives out of all real positives (*i.e.* true positives + false negatives). *Precision-recall curves* plot precision against recall when thresholding predictions at different values (*i.e.* varying the threshold defining positive vs. negative). See the Wikipedia article for "Precision and recall" and scikit-learn documentation for "Precision-Recall."

[44] These "grandmother cell"-like filters existed for PASCAL classes like bicycle, person, car, and cat.

[45] A *support vector machine (SVM)* is a kind of supervised machine learning algorithm. See the Wikipedia article for "Support vector machine."

[46] Alain et al., 2016 leverage a similar strategy of training linear classifiers on filter activations; however, Alain et al., 2016 focus on understanding how layer depth affected downstream task performance.

[47] However, Agrawal et al., 2014 also noted that for a few classes, such as person and bicycle, it appeared that only a few select filters were necessary to performance comparable to that acheived when using all filters.



(a) Precision-recall curves **(b)** # filters vs. accuracy

Figure 2.17: Quantifying the number of filters needed to encode a concept (Agrawal et al., 2014). **(a)** We show precision-recall curves that reflect performance when using individual filters to perform object detection for a specific concept (each line represents a filter). Because most curves drop immediately (with the exception of the "person" curve), this suggests that single filters typically are insufficient for detecting a concept consistently. **(b)** Here, object detection performance is plotted against the number of filters used to perform the task. How quickly the curves saturate (*i.e.* plateau) indicate how few filters are needed to encode a given concept. Image reproduced with permission from Agrawal et al., 2014.

found that 60% of filters correlated with specific concepts, Gonzalez-Garcia et al., 2016 found that 75% of filters did not systematically correlate with a semantic part. These differences may be attributed to the different methodologies used and/or different underlying datasets used to train the networks being evaluated.[48] By comparing clusters that grouped filter activations together with individual filters, J. Wang et al., 2015 show that their clusters acted as superior parts detectors when evaluated on detection tasks, thereby suggesting that semantic parts are better represented by populations of units rather than individual units.

Morcos et al., 2018 and Zhou et al., 2018c present opposing perspectives on the importance of class-selective hidden units. Morcos et al., 2018 argue that the more distributed a neural network representation is, the better it appears to generalize beyond the training distribution. They point out that techniques like dropout (Srivastava et al., 2014) and batch normalization (Ioffe et al., 2015),[49] which are known to improve model robustness, encourage relatively more distributed encodings. They also perform an ablation study in which they correlate the class selectivity of individual units with the impact they have on task performance when

[48] Gonzalez-Garcia et al., 2016 use the relatively smaller PASCAL-Part dataset (X. Chen et al., 2014), while Zhou et al., 2015 use the larger ImageNet (Russakovsky et al., 2015) and Places (Zhou et al., 2016b) datasets.

[49] Network dissection (Bau et al., 2017) highlight the effect dropout and batch normalization had on the distributed-ness of encodings. Bau et al., 2017's results affirm the batchnorm experiment of Morcos et al., 2018 yet disagree with the corresponding dropout experiment.

ablated. This experiment revealed a neutral or negative correlation between class selective and ablated performance, leading the authors to conclude that a unit's class selectivity is a poor predictor of its relevance to the downstream task.

In response, Zhou et al., 2018c highlight that, while class-selectivity of individiual units is a poor predictor of *overall* task performance, it is a strong predictor of class-specific performance. They redid the original ablation experiments as well as ran modified versions in which they found that ablating units that are highly selective for a given class negatively impacts the class-specific accuracy for that class.

2.3.3 Sensitivity to geometric transformations

Lastly, a number of works focus on characterizing properties of internal components of a model. In this section, we highlight those that focus on understanding a CNN representation's sensitivity to various geometric transformations (*e.g.* translation, scale, rotation) and how that changes based on the location of a representation in a network.

Zeiler et al., 2014 plots the change in probability of the predicted output class as an single image was systematically transformed.[50] Their qualitative results suggest that features of late CNN layers are relatively more invariant to geometric transformation than those of early layers.

Lenc et al., 2015 introduce a quantitative way to evaluate the equivariance, invariance, as well as equivalence of different representations.[51] They do this by systematically transforming the input, learning an additional "transformation" layer that "undoes" the input transformation at a given network depth, and quantifying changes to classification performance when a transformation layer is used.[52] One of their findings was that AlexNet (Krizhevsky et al., 2012) is largely invariant to horizontal flips and rescaling but is sensitive to vertical flips and 90-degree rotations. This makes sense because horizontal flipping and random crops, which typically include rescaling the image, are standard pre-processing steps when training a CNN. Furthermore, a lot of real-world objects have horizontal symmetry (*e.g.* an apple, a mug, etc.). Lenc et al., 2015 suggest that a CNN representation's sensitivity to

[50] They also do this another variant in which they plot the change in Euclidean distance of intermediate activations.

[51] *Equivariance* refers to how a representation changes as an input is transformed. *Invariance* refers to the lack of change (*i.e.* insensitivity) in a representation as an input is transformed. *Equivalence* refers to whether two different representations encode the same information (*i.e.* are functionally equivalent). These definitions are given by Lenc et al., 2015.

[52] Lenc et al., 2015 chose a linear layer to be the transformation layer.

different kinds of transformations is a reflection of whether those transformations are seen in training data (*i.e.* either naturally or due to pre-processing steps).

2.4 Interactive visualizations and visual interfaces

An *interactive visualization* is a visualization technique that leverages human interaction as an essential component of a visualization. Relatedly, an *interactive visual interface* is an interface that leverages human interaction to explore, navigate, and/or interact with multiple visualizations. The multidisciplinary field of *visual analytics* (Thomas et al., 2005) focuses on studying the use of such interfaces to facilitate analytical reasoning.[53]

While there is vast literature on visual analytics, in this section we primarily focus on works that develop interactive visualizations and visual interfaces for the purpose of understanding CNNs.[54]

Most research in this vein can be categorized along a few dimensions: First, what kind of *access* to the CNN does the visualization have? That is, does it visualize the internal components of a model (*i.e. "white-box"*) or focus on the relationship between inputs and outputs of a model (*i.e. "black-box"*). Second, at what *level* does it visualize? That is, does it visualize a particular *instance* (*i.e.* a single input to the model) or does it aim to characterize a *global*[55] component or view of a model (*e.g.* a hidden unit in the model, how a model performs on various subsets of data). Third, how *"reusable"* is this interactive visualization or interface? That is, does it only work for a particular model architecture or dataset (*e.g.* pedagogical tools that illustrate how a particular CNN functions) or can it be easily applied to a number of models trained on a variety of datasets? We primarily organize the rest of the section along the first and third dimensions (*e.g.* "white-box" vs. "black-box" and "reusability"). Figure 2.18 summarizes the works discussed in this section and categorizes them based on these dimensions.

[53] See the Wikipedia article on "Visual analytics."

[54] For a comprehensive survey on visual analytics for deep learning, refer to Hohman et al., 2018.

[55] A *global explanation* refers to an explanation of a model's behavior that is input independent (*i.e.* characterizes a model component and how it broadly behaves, not how it specifically behaves for a given input).

	Model				Data			Usage	Scope		Access		Interactivity	
	CNN	RNN	MLP	Other	Toy	Simple	Complex	Re-useable	Instance	Global	White-box	Black-box	Visualization	Interface
Abadi et al., 2016 (TensorBoard)	X	X	X	X	X	X	X	X		X	X	X		X
Bau et al., 2019 (GANPaint)	X						X		X			X		X
Carter et al, 2016		X					X		X		X		X	
Carter et al., 2019 (Activation Atlas)	X						X	X*		X	X		X	X
Fong et al., in prep (Similarity Overlays)	X						X	X	X		X		X	
Google PAIR, 2017 (Facets)				X			X	X		X		X		X
Harley, 2015	X					X			X		X		X	X
Hohman et al., 2019 (GAMut)				X			X		X	X		X		X
Hohman, 2017 (ShapeShop)	X		X			X				X	X			X
Kahng et al., 2018 (GANLab)			X		X					X	X			X
Madsen, 2019 (Memory in RNNs)		X					X					X	X	
Norton & Qi, 2017 (Adversarial Playground)	X					X			X		X			X
Olah et al., 2018 (Building Blocks)	X						X	X*	X	X	X		X	
Olah, 2014				X		X				X	X	X	X	
Smilkov et al., 2017 (TensorFlow Playground)			X		X					X	X			X
Strobelt et al., 2017 (LSTMVis)		X					X		X		X			X
Strobelt et al., 2018 (Seq2SeqVis)		X					X		X		X	X		X
Talbot et al., 2009 (EnsembleMatrix)				X			X			X		X		X
Torralba, 2017 (DrawNet)	X						X			X	X		X	X
Wattenberg et al., 2016				X	X					X				X
Webster, 2017 (Teachable Machines)	X						X	X	X			X		X
Wexler et al., 2019 (What-If)	X	X	X				X	X				X		X
Yosinski et al., 2015 (DeepVis)	X						X	X*	X	X	X			X
Zhou et al., 2018 (iForest)				X		X			X	X	X			X
Zhu et al., 2016	X						X		X			X		X

Figure 2.18: Summary of literature on interactive visualizations and visual interfaces. In this table, we summarize the works reviewed in this section and categorize them along a few dimensions, which are described at the beginning of section 2.4. * denotes that significant set-up and pre-computing is required to visualize a new model.

2.4.1 "White-box" visualizations

"White-box" tools aim to visualize an internal aspect of the model (*e.g.* filter activations, model parameters, model layers).

2.4.1.1 Non-reusable visualizations

Most such works are of a pedagogical nature: They enable deep exploration of a restricted set of models and/or datasets in order to gain intuition as to how a model works.

Visualizations of toy datasets. Several works focus on visualizing neural networks trained on toy datasets (*e.g.* classifying 2-dimensional (x, y) points into two classes). **TensorFlow Playground**[56] (Smilkov et al., 2017a) is an interactive visual interface that allows a user to control a wide range of aspects of training a *fully-connected* neural network[57] for classification or regression (*e.g.* the training datset, input features, number of hidden units and layers, and other *hyperparameters*) and visualizes the impact of these variables across training time (see fig. 2.19).

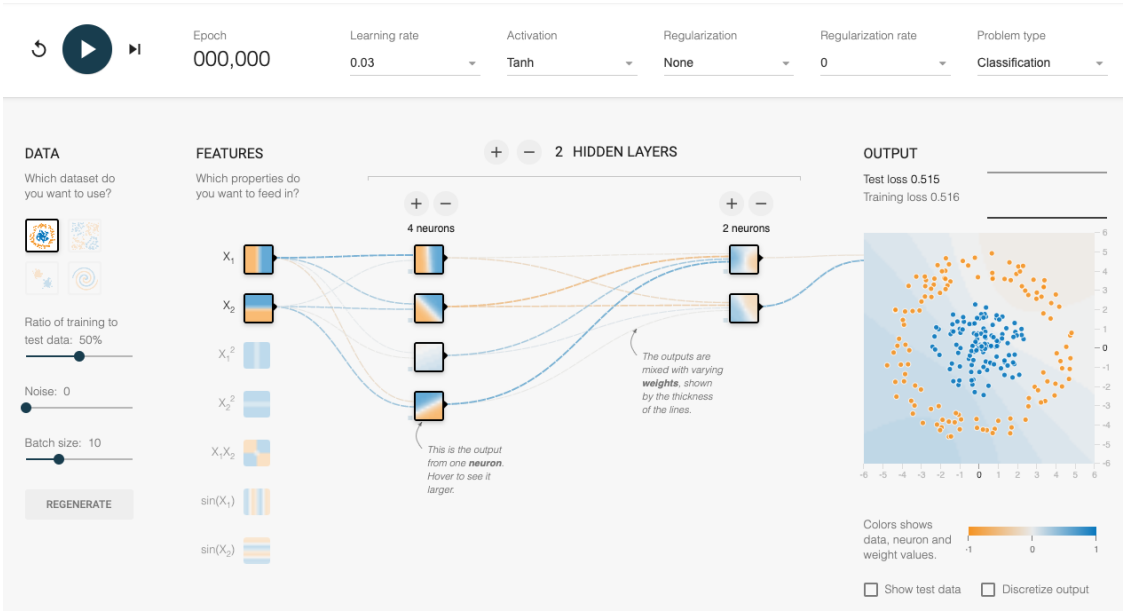


Figure 2.19: TensorFlow Playground (Smilkov et al., 2017a) In this visual interface, a user can control and inspect the training of a simple neural network – *i.e.* by selecting the training data (1st column), input features (2nd col), number of layers and neurons in each layer (3rd col), and other training hyperparameters (top). Images reproduced with permission of Smilkov et al., 2017a.

GANLab[58] (Kahng et al., 2018) is a similar tool that visualizes the impact of

[56] `https://playground.tensorflow.org`

[57] A fully-connected neural network does not contain convolutional layers that apply the same weights to many spatial neighborhoods but rather only consists of fully-connected, linear layers that apply a unique weight to every feature.

[58] `https://poloclub.github.io/ganlab`

design choices for *generative adversarial networks (GANs)*.[59] Wattenberg et al., 2016 present a similar interactive tool for understanding how the parameters for *t-SNE* plots (Maaten et al., 2008), which are frequently used to visualize high-dimensional data, affect their visualizations.

Visualizations of simple datasets. Other works visualize CNNs on simple image datasets, like the *MNIST* dataset (LeCun et al., 2010).[60] Inspired by TensorFlow Playground (Smilkov et al., 2017a), **Adversarial Playground** (Norton et al., 2017) allows the user to control the parameters of various techniques for generating *adversarial examples*[61] of MNIST images and visualizes the impact of those choices. Similarly, **ShapeShop** (Hohman et al., 2017) allows a user to design a training dataset from a set of basic shapes as well as a fully-connected or convolutional neural network model from a set of pre-defined options. It then visualizes a few predictions for the specified training set and model. It also allows a user to compare results between different experimental settings.

Harley, 2015 presents several instance-oriented interactive visualizations that allow a user to draw their own MNIST-like digit and interactively explore the intermediate activations and internal operations of a CNN network. Olah, 2014 presents a series of interactive visualizations that visualize the effects of applying several *dimensionality reduction*[62] techniques to MNIST examples.

Visualizations on complex datasets. Lastly, a number of works focus on visualizing the internal dynamics of popular, modern neural network models trained on more real-world examples (*e.g.* natural images, real-world corpus of text, diverse hand-writing samples). Although these provide deep insight into the models they visualize, they cannot be easily applied to novel models or datasets.

[59] *Generative adversarial networks (GANs)* are a machine learning paradigm in which two networks compete adversarially against one another. The discriminator network is optimized to distinguish real examples from fake, generated ones, while the generator network is optimized to fool the discriminator network such that it can't distinguish between real and fake examples. This paradigm encourages the generator network to produce realistic-looking generated examples. See the Wikipedia article for "Generative adversarial network" or I. Goodfellow et al., 2014 for more details.

[60] *MNIST* is a dataset that contains 28×28 black-and-white images of individual hand-written digits and is frequently used in a digits classification task.

[61] *Adversarial examples* (Szegedy et al., 2014) are maliciously generated images that contain nearly imperceptible modifications to real images yet are typically misclassified on models trained only on real examples.

[62] *Dimensionality reduction* refers to techniques that reduce the number of variables (*i.e.* dimensions) of high-dimensional data (*e.g.* transforming 28×28 MNIST examples to 2D points) by identifying the most important factors of variation. See the Wikipedia article on "Dimensionality reduction."

An interactive companion visualization for Cichy et al., 2016, **DrawNet** (Torralba, 2017) visualizes an AlexNet CNN (Krizhevsky et al., 2012) trained on ImageNet (Russakovsky et al., 2015), a large-scale object dataset. DrawNet allows a user to select a neuron in any of the 5 layers and visualizes the strongest connections going in and out of that neuron. Additionally, for several neurons with strong connections to the selected neuron, it shows 4 images, each with an receptive field overlay (Zhou et al., 2015) that most strongly activate the given neuron.

Several similar visualizations have been made for *recurrent neural networks (RNNs)*[63] trained on text and handwriting samples. **LSTMVis** (Strobelt et al., 2017) visualizes an *LSTM*[64] network and allows the user to explore hypotheses about the dynamics of its internal state and observe the effects on internal hidden units. It visualizes a number of LSTM models trained on real-world tasks and datasets (*e.g.* language models for text, music, and code; German-English translation; sentence summarization, etc.). Similarly, **Seq2Seq-Vis** (Strobelt et al., 2018) allows a user to explore a *sequence-to-sequence*[65] model at specific points (*i.e.* input or output to a network). Lastly, Carter et al., 2016 visualize a hand-writing prediction RNN and allows a user to dynamically input their own handwriting samples and simulataneously explore the internal activations of the model on their samples.

2.4.1.2 Reusable visualizations

In contrast to the previous works, a few works develop interactive visualizations and visual interfaces to be *reusable*, that is, to be used in the research and development process for a variety of novel models and datasets.

TensorFlow's (Abadi et al., 2016) **TensorBoard** provides a flexible, visual interface helps users monitor the training process and the model being trained by visualizing basic information as training progresses (*e.g.* plotting metrics, visualizing model graph (Wongsuphasawat et al., 2017), viewing distribution histograms of weights). Thanks to its API, it can be used easily to visualize custom objects (*e.g.* anything that takes the form of an image).

[63] *Recurrent neural networks (RNNs)* are another class of neural networks typically used for temporal data, such as text, as they are successively applied to temporal units in a sequence (*e.g.* characters in a sentence) and maintain an internal state (*i.e.* memory) to "remember" earlier units seen. See the Wikipedia article on "Recurrent neural network."

[64] An *LSTM (long short-term memory)* is a particular class of RNNs that can reason about longterm relationships in a temporal input sequence. See Olah, 2015's blog post on "Understanding LSTMs."

[65] A *sequence-to-sequence* model takes as input a sequence and also outputs a sequence (*e.g.* German-English translation, part-of-speech labelling, etc.).

The **Deep Visualization Toolbox (DeepVis)** (Yosinski et al., 2015) is explicitly oriented towards interpretability and allows a user to explore the internal representation of a Caffe (Jia et al., 2014) model. By combining several visualization techniques (*i.e.* activation maximization (Simonyan et al., 2014), DeConvNet (Zeiler et al., 2014), and top activated patches), the toolbox allows a user to present their own input image (*i.e.* via webcam), see the spatial activation patterns for all channels any layer, and zoom into a particular filter to see visualizations of what most activates it (see fig. 2.20).

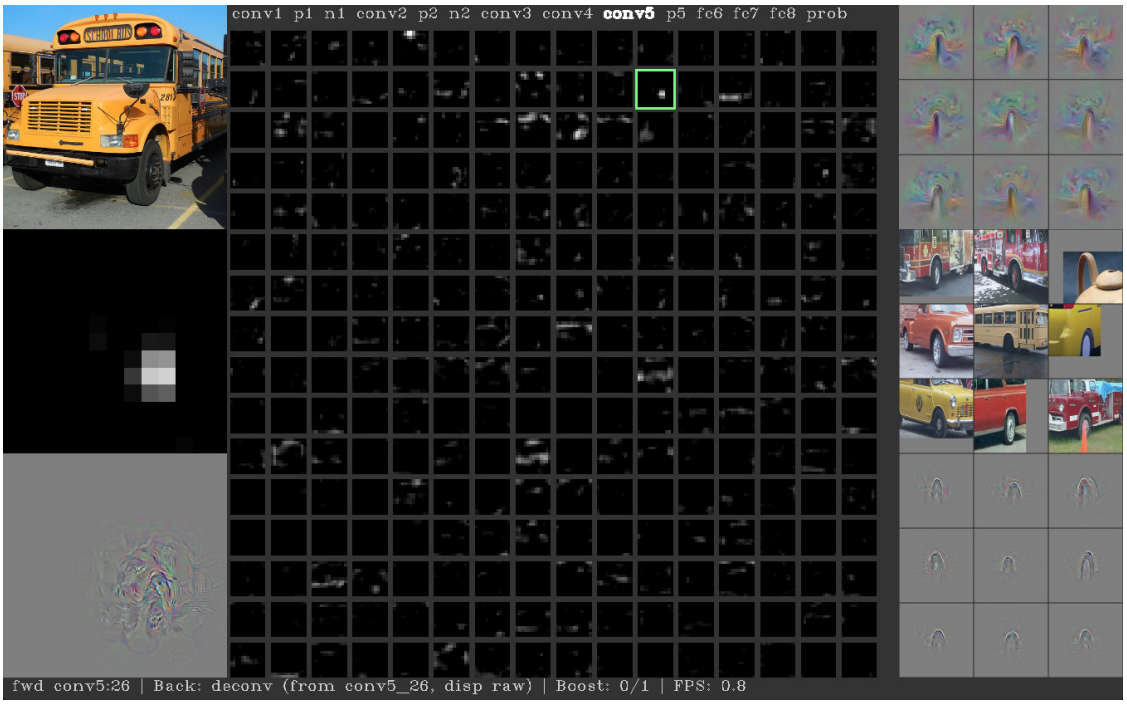


Figure 2.20: DeepVis (Yosinski et al., 2015). In this visual interface, a user can interactively explore the CNN representation of an image (*i.e.* school bus image) via a number of visualizations – *i.e.* individual filter activations (center and middle left), activation maximization (top right), top activated patches (middle right), and DeConvNet (bottom left and right). Users can select a layer (top center) and specific filter (green box) to focus on; they can also provide their own input via a webcam (not shown). Image reproduced with permission of Yosinski et al., 2015.

Leveraging the **Lucid**[66] interpretability library for TensorFlow models, **Building Blocks** (Olah et al., 2018) highlights how several visualization techniques (*e.g.* semantic dictionaries, activation grids, spatial and channel attribution, channel, and matrix factorization of activations) can be used together as building blocks to create novel visualizations that increase a user's understanding of a model. It uses

[66]https://github.com/tensorflow/lucid

interactivity to select a spatial region (*i.e.* an activation patch), convolutional filter, or factorized group for which to provide more specific visualizations.

In a similar vein, **Activation Atlas** (Carter et al., 2019) synthesizes two techniques: 1. feature visualizations (Olah et al., 2017) (see section 2.3.1.2); and 2. t-SNE plots (Maaten et al., 2008). By combining these visualizations, activation atlases visualize a wide range of CNN activations and how they relate to each other. A feature visualization typically visualizes a single direction in activation space, while t-SNE provides a global perspective of how data points relate to one another. Thus, by effectively showing a t-SNE of many feature visualizations and allowing the user to zoom in and out of the plot, activation atlases combine the depth of feature visualizations with the breadth of t-SNE plots.

Due to their reliance on optimized feature visualizations, Yosinski et al., 2015; Olah et al., 2018; Carter et al., 2019 require pre-computing feature visualizations for a novel model before being able to visualize it. The Lucid library also requires models to be imported as Lucid model objects. Thus, non-negligible set-up work is needed to use these tools for novel models.

2.4.2 "Black-box" visualizations

As mentioned earlier, "black-box" visualizations explain a model's behavior by reasoning about its inputs and outputs (*i.e.* without access to a model's inner workings).

2.4.2.1 Non-reuseable visualizations

Madsen, 2019 presents two interactive visualizations to demonstrate how memory works in RNNs: 1. a connectivity visualization allows the user to hover over characters in an input sentence and visually attributes previous characters that most influence the prediction for the selected character; and 2. an autocomplete visualization that allows a user to type in a free-form input and shows in real-time the top three predicted choices for the next word. Madsen, 2019 also combines these two visualizations by showing the three most probable words in the connectivity visualization as well as synchronized visualizations of the same input sequence being processed by different RNNs to enable model comparison.

2.4.2.2 Reuseable visualizations

Several works introduce easy-to-use interfaces that require minimal coding.

Facets (PAIR, 2017) provides an interactive visual interface for a user to easily explore their data by summarizing input features and their distributions

(*i.e.* Facets Overview) and by organizing many data points for interactive exploration (*i.e.* Facets Dive). Building off Facets, the **What-If** tool (Wexler et al., 2019) allows a user to inspect machine learning models. It provides support for comparing the performance of multiple models, visualizing model results, showing feature attributions for individual predictions, editing a data point, and more. Lastly, **Teachable Machines** (Webster, 2017) provides an easy-to-use interface that allows lay users to train their own ML models and evaluate them on user-provided inputs (*e.g.* interactively via a webcam).

2.4.3 Visualizations for other models

Although we primarily focus on interactive visualizations for discriminative[67] neural network models, we highlight a few notable works for other kinds of models here.

Generative adversarial networks (GANs). Zhu et al., 2016 present a human-computer interactive collaboration tool that empowers a user to transform colored scribbles into naturalistic images with the assistance of a GAN. Similarly, **GAN-Paint** (Bau et al., 2019a) allows a user to augment a natural image and add specific concepts (*e.g.* trees, bricks) with paint-like strokes.

Non-CNN models. Other visual analytic interfaces have been developed for other kinds of models, such as general additive models (*i.e.* **GAMut** (Hohman et al., 2019), random forest models (*i.e.* **iForest** (Zhao et al., 2018)) and ensembles of models (*i.e.* **EnsembleMatrix** (Talbot et al., 2009)).

[67] *e.g.* models trained for a predictive task.

3

Interpretable Explanations of Black Boxes by Meaningful Perturbation

The following paper was presented at the IEEE International Conference of Computer Vision (ICCV) at Venice, Italy in 2017 (Fong et al., 2017).[1,2]

[1]Supplementary materials can be found here: `http://openaccess.thecvf.com/content_ICCV_2017/supplemental/Fong_Interpretable_Explanations_of_ICCV_2017_supplemental.pdf`

[2]We expanded on the contents of this paper in a book chapter (Fong et al., 2019b).

Interpretable Explanations of Black Boxes by Meaningful Perturbation

Ruth C. Fong
University of Oxford
ruthfong@robots.ox.ac.uk

Andrea Vedaldi
University of Oxford
vedaldi@robots.ox.ac.uk

Abstract

As machine learning algorithms are increasingly applied to high impact yet high risk tasks, such as medical diagnosis or autonomous driving, it is critical that researchers can explain how such algorithms arrived at their predictions. In recent years, a number of image saliency methods have been developed to summarize where highly complex neural networks "look" in an image for evidence for their predictions. However, these techniques are limited by their heuristic nature and architectural constraints.

In this paper, we make two main contributions: First, we propose a general framework for learning different kinds of explanations for any black box algorithm. Second, we specialise the framework to find the part of an image most responsible for a classifier decision. Unlike previous works, our method is model-agnostic and testable because it is grounded in explicit and interpretable image perturbations.

1. Introduction

Given the powerful but often opaque nature of modern black box predictors such as deep neural networks [4, 5], there is a considerable interest in *explaining* and *understanding* predictors *a-posteriori*, after they have been learned. This remains largely an open problem. One reason is that we lack a formal understanding of what it means to explain a classifier. Most of the existing approaches [19, 16, 8, 7, 9, 19], etc., often produce intuitive visualizations; however, since such visualizations are primarily heuristic, their meaning remains unclear.

In this paper, we revisit the concept of "explanation" at a formal level, with the goal of developing principles and methods to explain any black box function f, e.g. a neural network object classifier. Since such a function is learned automatically from data, we would like to understand *what* f has learned to do and *how* it does it. Answering the "what" question means determining the properties of the map. The "how" question investigates the internal mechanisms that allow the map to achieve these properties. We focus mainly on the "what" question and argue that it can be answered by providing *interpretable rules* that describe the input-output relationship captured by f. For example, one rule could be that f is rotation invariant, in the sense that "$f(x) = f(x')$ whenever images x and x' are related by a rotation".

Figure 1. An example of a mask learned (right) by blurring an image (middle) to suppress the softmax probability of its target class (left: original image; softmax scores above images).

In this paper, we make several contributions. First, we propose the general framework of explanations as meta-predictors (sec. 2), extending [18]'s work. Second, we identify several pitfalls in designing automatic explanation systems. We show in particular that neural network artifacts are a major attractor for explanations. While artifacts are informative since they explain part of the network behavior, characterizing other properties of the network requires careful calibration of the *generality* and *interpretability* of explanations. Third, we reinterpret network saliency in our framework. We show that this provides a natural generalization of the gradient-based saliency technique of [15] by *integrating* information over several rounds of backpropagation in order to learn an explanation. We also compare this technique to other methods [15, 16, 20, 14, 19] in terms of their meaning and obtained results.

2. Related work

Our work builds on [15]'s gradient-based method, which backpropagates the gradient for a class label to the image layer. Other backpropagation methods include DeConvNet [19] and Guided Backprop [16, 8], which builds off of DeConvNet [19] and [15]'s gradient method to produce sharper visualizations.

Another set of techniques incorporate network activations into their visualizations: Class Activation Mapping

(CAM) [22] and its relaxed generalization Grad-CAM [14] visualize the linear combination of a late layer's activations and class-specific weights (or gradients for [14]), while Layer-Wise Relevance Propagation (LRP) [1] and Excitation Backprop [20] backpropagate an class-specific error signal though a network while multiplying it with each convolutional layer's activations.

With the exception of [15]'s gradient method, the above techniques introduce different backpropagation heuristics, which results in aesthetically pleasing but heuristic notions of image saliency. They also are not model-agnostic, with most being limited to neural networks (all except [15, 1]) and many requiring architectural modifications [19, 16, 8, 22] and/or access to intermediate layers [22, 14, 1, 20].

A few techniques examine the relationship between inputs and outputs by editing an input image and observing its effect on the output. These include greedily graying out segments of an image until it is misclassified [21] and visualizing the classification score drop when an image is occluded at fixed regions [19]. However, these techniques are limited by their approximate nature; we introduce a differentiable method that allows for the effect of the joint inclusion/exclusion of different image regions to be considered.

Our research also builds on the work of [18, 12, 2]. The idea of explanations as predictors is inspired by the work of [18], which we generalize to new types of explanations, from classification to invariance.

The Local Intepretable Model-Agnostic Explanation (LIME) framework [12] is relevant to our local explanation paradigm and saliency method (sections 3.2, 4) in that both use an function's output with respect to inputs from a neighborhood around an input x_0 that are generated by perturbing the image. However, their method takes much longer to converge ($N = 5000$ vs. our 300 iterations) and produces a coarse heatmap defined by fixed super-pixels.

Similar to how our paradigm aims to learn an image perturbation mask that minimizes a class score, feedback networks [2] learn gating masks after every ReLU in a network to maximize a class score. However, our masks are plainly interpretable as they directly edit the image while [2]'s ReLU gates are not and can not be directly used as a visual explanation; furthermore, their method requires architectural modification and may yield different results for different networks, while ours is model-agnostic.

3. Explaining black boxes with meta-learning

A *black box* is a map $f : \mathcal{X} \to \mathcal{Y}$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$, typically obtained from an opaque learning process. To make the discussion more concrete, consider as input color images $x : \Lambda \to \mathbb{R}^3$ where $\Lambda = \{1, \dots, H\} \times \{1, \dots, W\}$ is a discrete domain. The output $y \in \mathcal{Y}$ can be a boolean $\{-1, +1\}$ telling whether the image contains an object of a certain type (e.g. a *robin*), the probability of such an event, or some other interpretation of the image content.

3.1. Explanations as meta-predictors

An *explanation* is a rule that predicts the response of a black box f to certain inputs. For example, we can explain a behavior of a *robin* classifier by the rule $Q_1(x; f) = \{x \in \mathcal{X}_c \Leftrightarrow f(x) = +1\}$, where $\mathcal{X}_c \subset \mathcal{X}$ is the subset of all the *robin* images. Since f is imperfect, any such rule applies only approximately. We can measure the faithfulness of the explanation as its expected prediction error: $\mathcal{L}_1 = \mathbb{E}[1 - \delta_{Q_1(x;f)}]$, where δ_Q is the indicator function of event Q. Note that Q_1 implicitly requires a distribution $p(x)$ over possible images $\mathcal{X}$. Note also that $\mathcal{L}_1$ is simply the expected prediction error of the classifier. Unless we did not know that f was trained as a *robin* classifier, Q_1 is not very insightful, but it is interpretable since $\mathcal{X}_c$ is.

Explanations can also make relative statements about black box outcomes. For example, a black box f, could be rotation invariant: $Q_2(x, x'; f) = \{x \sim_{\text{rot}} x' \Rightarrow f(x) = f(x')\}$, where $x \sim_{\text{rot}} x'$ means that x and x' are related by a rotation. Just like before, we can measure the faithfulness of this explanation as $\mathcal{L}_2 = \mathbb{E}[1 - \delta_{Q_2(x,x';f)} | x \sim x']$.[1] This rule is interpretable because the relation $\sim_{\text{rot}}$ is.

Learning explanations. A significant advantage of formulating explanations as meta predictors is that their faithfulness can be measured as prediction accuracy. Furthermore, machine learning algorithms can be used to *discover explanations* automatically, by finding explanatory rules Q that apply to a certain classifier f out of a large pool of possible rules $\mathcal{Q}$.

In particular, finding the *most accurate* explanation Q is similar to a traditional learning problem and can be formulated computationally as a *regularized empirical risk minimization* such as:

$$\min_{Q \in \mathcal{Q}} \lambda \mathcal{R}(Q) + \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(Q(x_i; f), x_i, f), \; x_i \sim p(x). \quad (1)$$

Here, the regularizer $\mathcal{R}(Q)$ has two goals: to allow the explanation Q to generalize beyond the n samples $x_1, \dots, x_n$ considered in the optimization and to pick an explanation Q which is simple and thus, hopefully, more interpretable.

Maximally informative explanations. Simplicity and interpretability are often not sufficient to find good explanations and must be paired with informativeness. Consider the following variant of rule Q_2: $Q_3(x, x'; f, \theta) = \{x \sim_\theta x' \Rightarrow f(x) = f(x')\}$, where $x \sim_\theta x'$ means that x and x'

[1] For rotation invariance we condition on $x \sim x'$ because the probability of independently sampling rotated x and x' is zero, so that, without conditioning, Q_2 would be true with probability 1.

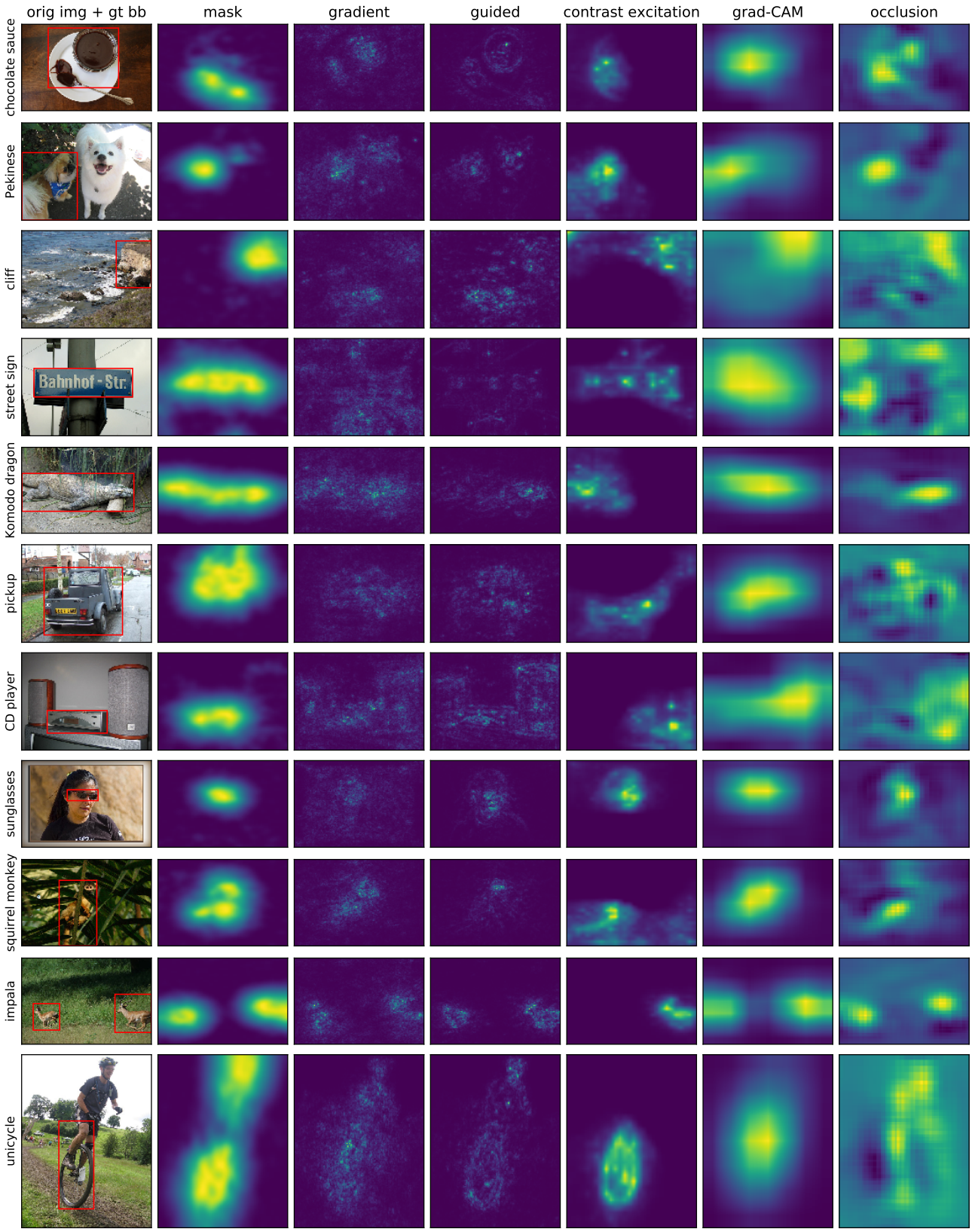


Figure 2. Comparison with other saliency methods. From left to right: original image with ground truth bounding box, learned mask subtracted from 1 (our method), gradient-based saliency [15], guided backprop [16, 8], contrastive excitation backprop [20], Grad-CAM [14], and occlusion [19].

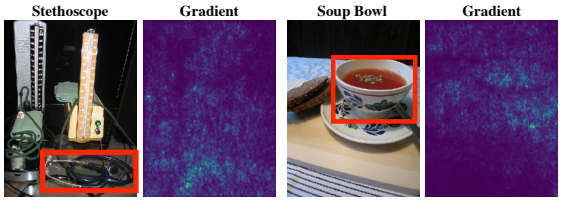


Figure 3. Gradient saliency maps of [15]. A red bounding box highlight the object which is meant to be recognized in the image. Note the strong response in apparently non-relevant image regions.

are related by a rotation of an angle $\leq \theta$. Explanations for larger angles imply the ones for smaller ones, with $\theta = 0$ being trivially satisfied. The regularizer $\mathcal{R}(Q_3(\cdot;\theta)) = -\theta$ can then be used to select a maximal angle and thus find an explanation that is as informative as possible.[2]

3.2. Local explanations

A *local explanation* is a rule $Q(x; f, x_0)$ that predicts the response of f in a neighborhood of a certain point x_0. If f is smooth at x_0, it is natural to construct Q by using the first-order Taylor expansion of f:

$$f(x) \approx Q(x; f, x_0) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle. \quad (2)$$

This formulation provides an interpretation of [15]'s saliency maps, which visualize the gradient $S_1(x_0) = \nabla f(x_0)$ as an indication of salient image regions. They argue that large values of the gradient identify pixels that strongly affect the network output. However, an issue is that this interpretation *breaks for a linear classifier*: If $f(x) = \langle w, x \rangle + b$, $S_1(x_0) = \nabla f(x_0) = w$ is independent of the image x_0 and hence cannot be interpreted as saliency.

The reason for this failure is that eq. (2) studies the variation of f for arbitrary displacements $\Delta_x = x - x_0$ from x_0 and, for a linear classifier, the change is the same regardless of the starting point x_0. For a non-linear black box f such as a neural network, this problem is reduced but not eliminated, and can explain why the saliency map S_1 is rather diffuse, with strong responses even where no obvious information can be found in the image (fig. 3).

We argue that the meaning of explanations depends in large part on the *meaning of varying the input x to the black box*. For example, explanations in sec. 3.1 are based on letting x vary in image category or in rotation. For saliency, one is interested in finding image regions that impact f's output. Thus, it is natural to consider perturbations x obtained by deleting subregions of x_0. If we model deletion by multiplying x_0 point-wise by a mask m,

[2] Naively, strict invariance for any $\theta > 0$ implies invariance to arbitrary rotations as small rotations compose into larger ones. However, the formulation can still be used to describe rotation insensitivity (when f varies slowly with rotation), or $\sim_\theta$'s meaning can be changed to indicate rotation w.r.t. a canonical "upright" direction for a certain object classes, etc.

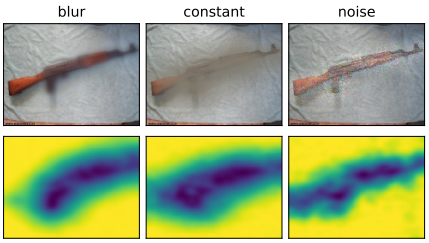


Figure 4. Perturbation types. Bottom: perturbation mask; top: effect of blur, constant, and noise perturbations.

this amounts to studying the function $f(x_0 \odot m)$[3]. The Taylor expansion of f at $m = (1, 1, \ldots, 1)$ is $S_2(x_0) = df(x_0 \odot m)/dm|_{m=(1,\ldots,1)} = \nabla f(x_0) \odot x_0$. For a linear classifier f, this results in the saliency $S_2(x_0) = w \odot x_0$, which is large for pixels for which x_0 and w are large simultaneously. We refine this idea for non-linear classifiers in the next section.

4. Saliency revisited

4.1. Meaningful image perturbations

In order to define an explanatory rule for a black box $f(x)$, one must start by specifying which variations of the input x will be used to study f. The aim of saliency is to identify which regions of an image x_0 are used by the black box to produce the output value $f(x_0)$. We can do so by observing how the value of $f(x)$ changes as x is obtained "deleting" different regions R of x_0. For example, if $f(x_0) = +1$ denotes a *robin* image, we expect that $f(x) = +1$ as well unless the choice of R deletes the robin from the image. Given that x is a perturbation of x_0, this is a local explanation (sec. 3.2) and we expect the explanation to characterize the relationship between f and x_0.

While conceptually simple, there are several problems with this idea. The first one is to specify what it means "delete" information. As discussed in detail in sec. 4.3, we are generally interested in simulating naturalistic or plausible imaging effect, leading to more meaningful perturbations and hence explanations. Since we do not have access to the image generation process, we consider three obvious proxies: replacing the region R with a constant value, injecting noise, and blurring the image (fig. 4).

Formally, let $m : \Lambda \to [0, 1]$ be a *mask*, associating each pixel $u \in \Lambda$ with a scalar value $m(u)$. Then the perturbation operator is defined as

$$[\Phi(x_0; m)](u) = \begin{cases} m(u)x_0(u) + (1 - m(u))\mu_0, & \text{constant}, \\ m(u)x_0(u) + (1 - m(u))\eta(u), & \text{noise}, \\ \int g_{\sigma_0 m(u)}(v - u)x_0(v)\, dv, & \text{blur}, \end{cases}$$

where μ_0 is an average color, $\eta(u)$ are i.i.d. Gaussian noise samples for each pixel and σ_0 is the maximum isotropic

[3] $\odot$ is the Hadamard or element-wise product of vectors.

standard deviation of the Gaussian blur kernel g_σ (we use $\sigma_0 = 10$, which yields a significantly blurred image).

4.2. Deletion and preservation

Given an image x_0, our goal is to summarize compactly the effect of deleting image regions in order to explain the behavior of the black box. One approach to this problem is to find deletion regions that are maximally informative.

In order to simplify the discussion, in the rest of the paper we consider black boxes $f(x) \in \mathbb{R}^C$ that generate a vector of scores for different hypotheses about the content of the image (e.g. as a softmax probability layer in a neural network). Then, we consider a "deletion game" where the goal is to find the smallest deletion mask m that causes the score $f_c(\Phi(x_0; m)) \ll f_c(x_0)$ to drop significantly, where c is the target class. Finding m can be formulated as the following learning problem:

$$m^* = \operatorname*{argmin}_{m \in [0,1]^\Lambda} \lambda \|\mathbf{1} - m\|_1 + f_c(\Phi(x_0; m)) \qquad (3)$$

where λ encourages most of the mask to be turned off (hence deleting a small subset of x_0). In this manner, we can find a highly informative region for the network.

One can also play an symmetric "preservation game", where the goal is to find the smallest subset of the image that must be retained to preserve the score $f_c(\Phi(x_0; m)) \geq f_c(x_0)$: $m^* = \operatorname{argmin}_m \lambda \|m\|_1 - f_c(\Phi(x_0; m))$. The main difference is that the deletion game removes enough evidence to prevent the network from recognizing the object in the image, whereas the preservation game finds a minimal subset of sufficient evidence.

Iterated gradients. Both optimization problems are solved by using a local search by means of gradient descent methods. In this manner, our method extracts information from the black box f by computing its gradient, similar to the approach of [15]. However, it differs in that it extracts this information progressively, over several gradient evaluations, accumulating increasingly more information over time.

4.3. Dealing with artifacts

By committing to finding a single representative perturbation, our approach incurs the risk of triggering artifacts of the black box. Neural networks, in particular, are known to be affected by surprising artifacts [5, 10, 7]; these works demonstrate that it is possible to find particular inputs that can drive the neural network to generate nonsensical or unexpected outputs. This is not entirely surprising since neural networks are trained discriminatively on natural image statistics. While not all artifacts look "unnatural", nevertheless they form a subset of images that is sampled with negligible probability when the network is operated normally.

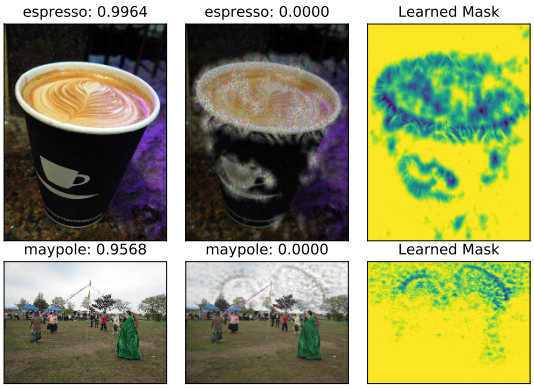


Figure 5. From left to right: an image correctly classified with large confidence by GoogLeNet [17]; a perturbed image that is not recognized correctly anymore; the deletion mask learned with artifacts. Top: A mask learned by minimizing the top five predicted classes by jointly applying the constant, random noise, and blur perturbations. Note that the mask learns to add highly structured swirls along the rim of the cup ($\gamma = 1, \lambda_1 = 10^{-5}, \lambda_2 = 10^{-3}, \beta = 3$). Bottom: A minimizing-top5 mask learned by applying a constant perturbation. Notice that the mask learns to introduce sharp, unnatural artifacts in the sky instead of deleting the pole ($\gamma = 0.1, \lambda_1 = 10^{-4}, \lambda_2 = 10^{-2}, \beta = 3$).

Although the existence and characterization of artifacts is an interesting problem *per se*, we wish to characterize the behavior of black boxes under normal operating conditions. Unfortunately, as illustrated in fig. 5, objectives such as eq. (3) are strongly attracted by such artifacts, and naively learn subtly-structured deletion masks that trigger them. This is particularly true for the noise and constant perturbations as they can more easily than blur create artifacts using sharp color contrasts (fig. 5, bottom row).

We suggests two approaches to avoid such artifacts in generating explanations. The first one is that powerful explanations should, just like any predictor, generalize as much as possible. For the deletion game, this means not relying on the details of a singly-learned mask m. Hence, we reformulate the problem to apply the mask m stochastically, up to small random jitter.

Second, we argue that masks co-adapted with network artifacts are *not representative of natural perturbations*. As noted before, the meaning of an explanation depends on the meaning of the changes applied to the input x; to obtain a mask more representative of natural perturbations we can encourage it to have a simple, regular structure which cannot be co-adapted to artifacts. We do so by regularizing m in total-variation (TV) norm and upsampling it from a low resolution version.

With these two modifications, eq. (3) becomes:

$$\min_{m \in [0,1]^\Lambda} \lambda_1 \|\mathbf{1} - m\|_1 + \lambda_2 \sum_{u \in \Lambda} \|\nabla m(u)\|_\beta^\beta + \mathbb{E}_\tau[f_c(\Phi(x_0(\cdot - \tau), m))], \qquad (4)$$

Figure 6. Interrogating suppressive effects. Left to right: original image with the learned mask overlaid; a boxed perturbation chosen out of interest (the truck's middle bounding box was chosen based on the contrastive excitation backprop heatmap from fig. 2, row 6); another boxed perturbation based on the learned mask (target softmax probabilities of for the original and perturbed images are listed above).

where $M(v) = \sum_u g_{\sigma_m}(v/s - u)m(u)$. is the upsampled mask and g_{σ_m} is a 2D Gaussian kernel. Equation (4) can be optimized using stochastic gradient descent.

Implementation details. Unless otherwise specified, the visualizations shown were generated using Adam [3] to minimize GoogLeNet's [17] softmax probability of the target class by using the blur perturbation with the following parameters: learning rate $\gamma = 0.1, N = 300$ iterations, $\lambda_1 = 10^{-4}, \lambda_2 = 10^{-2}, \beta = 3$, upsampling a mask ($28 \times 28$ for GoogLeNet) by a factor of $\delta = 8$, blurring the upsampled mask with $g_{\sigma_m=5}$, and jittering the mask by drawing an integer from the discrete uniform distribution on $[0, \tau)$ where $\tau = 4$. We initialize the mask as the smallest centered circular mask that suppresses the score of the original image by 99% when compared to that of the fully perturbed image, i.e. a fully blurred image.

5. Experiments

5.1. Interpretability

An advantage of the proposed framework is that the generated visualizations are clearly interpretable. For example, the deletion game produces a minimal mask that prevents the network from recognizing the object.

When compared to other techniques (fig. 2), this method can pinpoint the reason why a certain object is recognized without highlighting non-essential evidence. This can be noted in fig. 2 for the CD player (row 7) where other visualizations also emphasize the neighboring speakers, and similarly for the cliff (row 3), the street sign (row 4), and the sunglasses (row 8). Sometimes this shows that only a part of an object is essential: the face of the Pekenese dog (row 2), the upper half of the truck (row 6), and the spoon on the chocolate sauce plate (row 1) are all found to be minimally sufficient parts.

While contrastive excitation backprop generated heatmaps that were most similar to our masks, our method introduces a quantitative criterion (i.e., maximally suppressing a target class score), and its verifiable nature (i.e., direct edits to an image), allows us to compare differing proposed saliency explanations and demonstrate that our learned masks are better on this metric. In fig. 6, row 2, we show that applying a bounded perturbation informed by our learned mask significantly suppresses the truck softmax score, whereas a boxed perturbation on the truck's back bumper, which is highlighted by contrastive excitation backprop in fig. 2, row 6, actually increases the score from 0.717 to 0.850.

The principled interpretability of our method also allows us to identify instances when an algorithm may have learned the wrong association. In the case of the chocolate sauce in fig. 6, row 1, it is surprising that the spoon is highlighted by our learned mask, as one might expect the sauce-filled jar to be more salient. However, manually perturbing the image reveals that indeed the spoon is more suppressive than the jar. One explanation is that the ImageNet "chocolate sauce" images contain more spoons than jars, which appears to be true upon examining some images. More generally, our method allows us to diagnose highly-predictive yet non-intuitive and possibly misleading correlations by identified machine learning algorithms in the data.

5.2. Deletion region representativeness

To test that our learned masks are generalizable and robust against artifacts, we simplify our masks by further blurring them and then slicing them into binary masks by thresholding the smoothed masks by $\alpha \in [0 : 0.05 : 0.95]$ (fig. 7, top; $\alpha \in [0.2, 0.6]$ tends to cover the salient part identified by the learned mask). We then use these simplified masks to edit a set of 5,000 ImageNet images with constant, noise, and blur perturbations. Using GoogLeNet [17], we compute normalized softmax probabilities[4] (fig. 7, bottom). The fact that these simplified masks quickly suppress scores as α increases for all three perturbations gives confidence that the learned masks are identifying the right regions to perturb and are generalizable to a set of extracted masks and other perturbations that they were not trained on.

5.3. Minimality of deletions

In this experiments we assess the ability of our method to correctly identify a minimal region that suppresses the object. Given the output saliency map, we normalize its intensities to lie in the range $[0, 1]$, threshold it with $h \in [0 : 0.1 : 1]$, and fit the tightest bounding box around the resulting heatmap. We then blur the image in the box and compute the normalized[4] target softmax probability from

[4] $p' = \frac{p - p_0}{p_0 - p_b}$, where p, p_0, p_b are the masked, original, and fully blurred images' scores

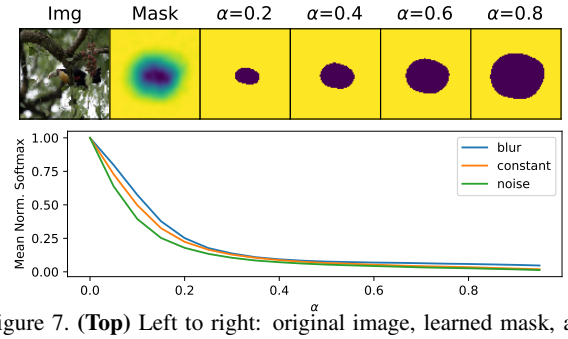


Figure 7. **(Top)** Left to right: original image, learned mask, and simplified masks for sec. 5.2 (not shown: further smoothed mask). **(Bottom)** Swift softmax score suppression is observed when using all three perturbations with simplified binary masks (top) derived from our learned masks, thereby showing the generality of our masks.

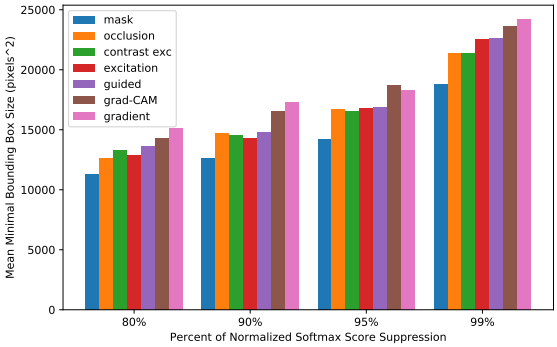


Figure 8. On average, our method generates the smallest bounding boxes that, when used to blur the original images, highly suppress their normalized softmax probabilities (standard error included).

GoogLeNet [17] of the partially blurred image.

From these bounding boxes and normalized scores, for a given amount of score suppression, we find the smallest bounding box that achieves that amount of suppression. Figure 8 shows that, on average, our method yields the smallest minimal bounding boxes when considering suppressive effects of $80\%, 90\%, 95\%,$ and 99%. These results show that our method finds a small salient area that strongly impacts the network.

5.4. Testing hypotheses: animal part saliency

From qualitatively examining learned masks for different animal images, we noticed that faces appeared to be more salient than appendages like feet. Because we produce dense heatmaps, we can test this hypothesis. From an annotated subset of the ImageNet dataset that identifies the keypoint locations of non-occluded eyes and feet of vertebrate animals [11], we select images from classes that have at least 10 images which each contain at least one eye and foot annotation, resulting in a set of 3558 images from 76 animal classes (fig. 9). For every keypoint, we calculate the average heatmap intensity of a 5×5 window around the keypoint. For all 76 classes, the mean average intensity of eyes were lower and thus more salient than that of feet (see supplementary materials for class-specific results).

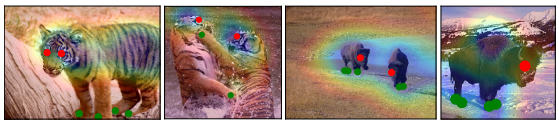


Figure 9. "tiger" (left two) and "bison" (right two) images with eyes and feet annotations from [11]; our learned masks are overlaid. The mean average feet:eyes intensity ratio for "tigers" ($N = 25$) is 3.82, while that for bisons ($N = 22$) is 1.07.

5.5. Adversarial defense

Adversarial examples [5] are often generated using a complementary optimization procedure to our method that learns a imperceptible pattern of noise which causes an image to be misclassified when added to it. Using our re-implementation of the highly effective one-step iterative method ($\epsilon = 8$) [5] to generate adversarial examples, our method yielded visually distinct, abnormal masks compared to those produced on natural images (fig. 10, left). We train an Alexnet [4] classifier (learning rate $\lambda_{lr} = 10^{-2}$, weight decay $\lambda_{L1} = 10^{-4}$, and momentum $\gamma = 0.9$) to distinguish between clean and adversarial images by using a given heatmap visualization with respect to the top predicted class on the clean and adversarial images (fig. 10, right); our method greatly outperforms the other methods and achieves a discriminating accuracy of 93.6%.

Lastly, when our learned masks are applied back to their corresponding adversarial images, they not only minimize the adversarial label but often allow the original, predicted label from the clean image to rise back as the top predicted class. Our method recovers the original label predicted on the clean image 40.64% of time and the ground truth label 37.32% ($N = 5000$). Moreover, 100% of the time the original, predicted label was recovered as one of top-5 predicted labels in the "mask+adversarial" setting. To our knowledge, this is the first work that is able to recover originally predicted labels without any modification to the training set-up and/or network architecture.

5.6. Localization and pointing

Saliency methods are often assessed in terms of weakly-supervised localization and a pointing game [20], which tests how discriminative a heatmap method is by calculating the precision with which a heatmap's maximum point lies on an instance of a given object class, for more harder datasets like COCO [6]. Because the deletion game is meant to discover minimal salient part and/or spurious correlation, we do not expect it to be particularly competitive on localization and pointing but tested them for completeness.

For localization, similar to [20, 2], we predict a bounding box for the most dominant object in each of ∼50k

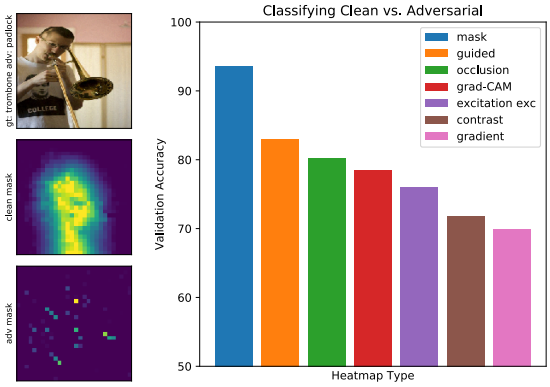


Figure 10. **(Left)** Difference between learned masks for clean (middle) and adversarial (bottom) images (28×28 masks shown without bilinear upsampling). **(Right)** Classification accuracy for discriminating between clean vs. adversarial images using heatmap visualizations ($N_{trn} = 4000, N_{val} = 1000$).

ImageNet [13] validation images and employ three simple thresholding methods for fitting bounding boxes. First, for value thresholding, we normalize heatmaps to be in the range of $[0, 1]$ and then threshold them by their value with $\alpha \in [0 : 0.05 : 0.95]$. Second, for energy thresholding [2], we threshold heatmaps by the percentage of energy their most salient subset covered with $\alpha \in [0 : 0.05 : 0.95]$. Finally, with mean thresholding [20], we threshold a heatmap by $\tau = \alpha\mu_I$, where μ_I is the mean intensity of the heatmap and $\alpha \in [0 : 0.5 : 10]$. For each thresholding method, we search for the optimal α value on a heldout set. Localization error was calculated as the IOU with a threshold of 0.5.

Table 1 confirms that our method performs reasonably and shows that the three thresholding techniques affect each method differently. Non-contrastive, excitation backprop [20] performs best when using energy and mean thresholding; however, our method performs best with value thresholding and is competitive when using the other methods: It beats gradient [15] and guided backprop [16] when using energy thresholding; beats LRP [1], CAM [22], and contrastive excitation backprop [20] when using mean thresholding (recall from fig. 2 that the contrastive method is visually most similar to mask); and out-performs Grad-CAM [14] and occlusion [19] for all thresholding methods.

For pointing, table 2 shows that our method outperforms the center baseline, gradient, and guided backprop methods and beats Grad-CAM on the set of difficult images (images for which 1) the total area of the target category is less than 25% of the image and 2) there are at least two different object classes). We noticed qualitatively that our method did not produce salient heatmaps when objects were very small. This is due to L1 and TV regularization, which yield well-formed masks for easily visible objects. We test two variants of occlusion [19], blur and variable occlusion, to interrogate if 1) the blur perturbation with smoothed masks

	Val-α*	Err (%)	Ene-α*	Err	Mea-α*	Err
Grad [15]	0.25	46.0	0.10	43.9	5.0	41.7[§]
Guid [16, 8]	0.05	50.2	0.30	47.0	4.5	42.0[§]
Exc [20]	0.15	46.1	0.60	**38.7**	1.5	**39.0**[§]
C Exc [20]	—	—	—	—	0.0	57.0[†]
Feed [2]	—	—	0.95	38.8[†]	—	—
LRP [1]	—	—	—	—	1.0	57.8[†]
CAM [22]	—	—	—	—	1.0	48.1[†]
Grad-CAM [14]	0.30	48.1	0.70	48.0	1.0	47.5
Occlusion [19]	0.30	51.2	0.55	49.4	1.0	48.6
Mask[‡]	0.10	**44.0**	0.95	43.1	0.5	43.2

Table 1. Optimal α thresholds and error rates from the weak localization task on the ImageNet validation set using saliency heatmaps to generate bounding boxes. [†]Feedback error rate are taken from [2]; all others (contrastive excitation BP, LRP, and CAM) are taken from [20]. [§]Using [20]'s code, we recalculated these errors, which are $\leq 0.4\%$ of the originally reported rates. [‡]Minimized top5 predicted classes' softmax scores and used $\lambda_1 = 10^{-3}$ and $\beta = 2.0$ (examples in supplementary materials).

	Ctr	Grad	Guid	Exc	CExc	G-CAM	Occ	Occ[§]	V-Occ[†]	Mask[‡]
All	27.93	36.40	32.68	41.78	**50.95**	41.10	44.50	45.41	42.31	37.49
Diff	17.86	28.21	26.16	32.73	**41.99**	30.59	36.45	37.45	33.87	30.64

Table 2. Pointing Game [20] Precision on COCO Val Subset ($N \approx$ 20k). [§]Occluded with circles ($r = 35/2$) softened by $g_{\sigma_m=10}$ and used to perturb with blur ($\sigma = 10$). [†]Occluded with variable-sized blur circles; from the top 10% most suppressive occlusions, the one with the smallest radius is chosen and its center is used as the point. [‡]Used min. top-5 hyper-parameters ($\lambda_1 = 10^{-3}$, $\beta = 2.0$).

is most effective, and 2) using the smallest, highly suppressive mask is sufficient (Occ[§] and V-Occ in table 2 respectively). Blur occlusion outperforms all methods except contrast excitation backprop while variable while variable occlusion outperforms all except contrast excitation backprop and the other occlusion methods, suggesting that our perturbation choice of blur and principle of identifying the smallest, highly suppressive mask is sound even if our implementation struggles on this task (see supplementary materials for examples and implementation details).

6. Conclusions

We propose a comprehensive, formal framework for learning explanations as meta-predictors. We also present a novel image saliency paradigm that learns *where* an algorithm *looks* by discovering which parts of an image most affect its output score when perturbed. Unlike many saliency techniques, our method explicitly edits to the image, making it interpretable and testable. We demonstrate numerous applications of our method, and contribute new insights into the fragility of neural networks and their susceptibility to artifacts.

References

[1] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 2, 8

[2] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2956–2964, 2015. 2, 7, 8

[3] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 7

[5] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016. 1, 5, 7

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 7

[7] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015. 1, 5

[8] A. Mahendran and A. Vedaldi. Salient deconvolutional networks. In *European Conference on Computer Vision*, pages 120–135. Springer International Publishing, 2016. 1, 2, 3, 8

[9] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016. 1

[10] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015. 5

[11] D. Novotny, D. Larlus, and A. Vedaldi. I have seen enough: Transferring parts across categories. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016. 7

[12] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. 2

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 8

[14] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*, 2016. 1, 2, 3, 8

[15] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. ICLR*, 2014. 1, 2, 3, 4, 5, 8

[16] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 1, 2, 3, 8

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 5, 6, 7

[18] R. Turner. A model explanation system. In *Proc. NIPS Workshop on Black Box Learning and Inference*, 2015. 1, 2

[19] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 1, 2, 3, 8

[20] J. Zhang, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff. Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, pages 543–559. Springer, 2016. 1, 2, 3, 7, 8

[21] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014. 2

[22] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016. 2, 8

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Interpretable Explanations of Black Boxes by Meaningful Perturbation
Publication Status	Published
Publication Details	Ruth C. Fong and Andrea Vedaldi. Interpretable Explanations of Black Box Algorithms by Meaningful Perturbation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

Student Confirmation

Student Name:	RUTH FONG		
Contribution to the Paper	A.V. proposed the initial idea. R.C.F. developed the idea into a working algorithm, ran experiments, and generated figures. R.C.F. and A.V. designed research and wrote the paper text.		
Signature		Date	6 April 2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: ANDREA VEDALDI		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

4

Understanding Deep Networks via Extremal Perturbations and Smooth Masks

The following paper was presented as an *oral* presentation at the IEEE International Conference on Computer Vision (ICCV) at Seoul, South Korea in 2019 (Fong et al., 2019a).[1]

[1]Supplementary materials can be found here: `http://ruthcfong.github.io/files/fong19_extremal_supps.pdf`

Understanding Deep Networks via Extremal Perturbations and Smooth Masks

Ruth Fong[†]
University of Oxford[*]

Mandela Patrick[†]
University of Oxford

Andrea Vedaldi
Facebook AI Research

Abstract

The problem of attribution is concerned with identifying the parts of an input that are responsible for a model's output. An important family of attribution methods is based on measuring the effect of perturbations applied to the input.In this paper, we discuss some of the shortcomings of existing approaches to perturbation analysis and address them by introducing the concept of extremal perturbations, which are theoretically grounded and interpretable. We also introduce a number of technical innovations to compute extremal perturbations, including a new area constraint and a parametric family of smooth perturbations, which allow us to remove all tunable hyper-parameters from the optimization problem. We analyze the effect of perturbations as a function of their area, demonstrating excellent sensitivity to the spatial properties of the deep neural network under stimulation. We also extend perturbation analysis to the intermediate layers of a network. This application allows us to identify the salient channels necessary for classification, which, when visualized using feature inversion, can be used to elucidate model behavior.

1. Introduction

Deep networks often have excellent prediction accuracy, but the basis of their predictions is usually difficult to understand. *Attribution* aims at characterising the response of neural networks by finding which parts of the network's input are the most responsible for determining its output. Most attribution methods are based on backtracking the network's activations from the output back to the input, usually via a modification of the backpropagation algorithm [22, 30, 25, 31, 21, 2]. When applied to computer vision models, these methods result in *saliency maps* that highlight important regions in the input image.

However, most attribution methods do not start from a definition of what makes an input region important for the neural network. Instead, most saliency maps are validated *a-posteriori* by either showing that they correlate with the image content (e.g., by highlighting relevant object categories), or that they find image regions that, if perturbed, have a large effect on the network's output (see Sec. 2).

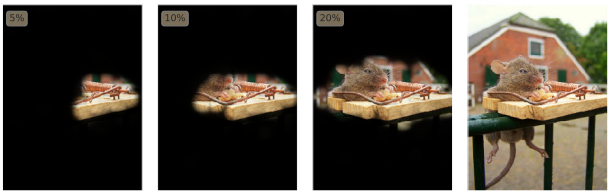


Figure 1: **Extremal perturbations** are regions of an image that, for a given area (boxed), maximally affect the activation of a certain neuron in a neural network (i.e., "mousetrap" class score). As the area of the perturbation is increased, the method reveals more of the image, in order of decreasing importance. For clarity, we black out the masked regions; in practice, the network sees blurred regions.

Some attribution methods, on the other hand, directly perform an analysis of the effect of *perturbing* the network's input on its output [30, 19, 6, 4]. This usually amounts to selectively deleting (or preserving) parts of the input and observing the effect of that change to the model's output. The advantage is that the meaning of such an analysis is clear from the outset. However, this is not as straightforward as it may seem on a first glance. First, since it is not possible to visualise *all* possible perturbations, one must find *representative* ones. Since larger perturbations will have, on average, a larger effect on the network, one is usually interested in small perturbations with a large effect (or large perturbations with a small effect). Second, Fong and Vedaldi [6] show that searching for perturbations with a large effect on the network's output usually results in *pathological* perturbations that trigger adversarial effects in the network. Characterizing instead the *typical* behavior of the model requires restricting the search to more representative perturbations via regularization terms. This results in an optimization problem that trades off maximizing the effect of the perturbation with its smoothness and size. In practice, this trade off is difficult to control numerically and somewhat obscures the meaning of the analysis.

In this paper, we make three contributions. First, instead of mixing several effects in a single energy term to optimize

[*]Work done as a contractor in FAIR. [†] denotes equal contributions.

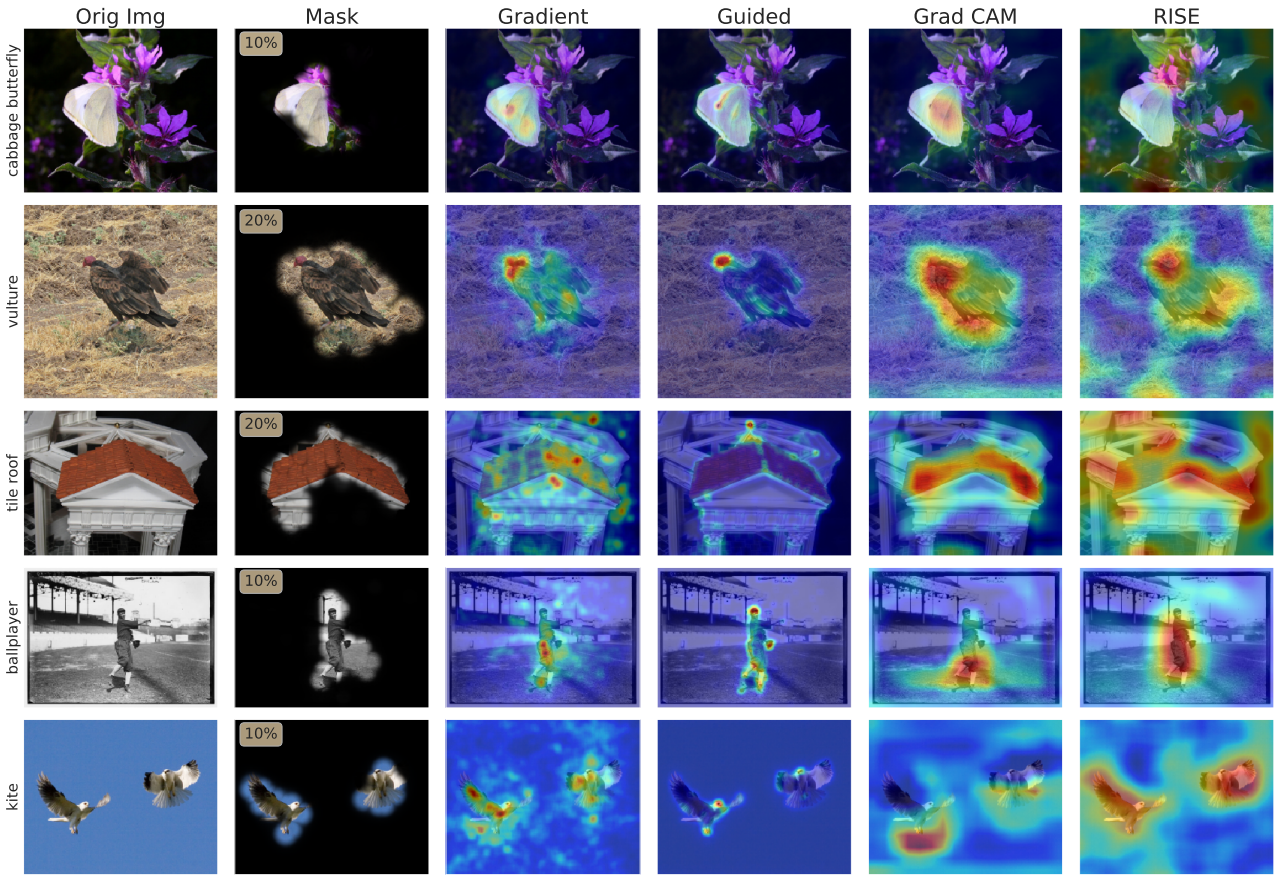


Figure 2: **Comparison with other attribution methods.** We compare our extremal perturbations (optimal area a^* in box) to several popular attribution methods: gradient [22], guided backpropagation [25], Grad-CAM [21], and RISE [19].

as in Fong and Vedaldi [6], we introduce the concept of *extremal perturbations*. A perturbation is extremal if it has maximal effect on the network's output among all perturbations of a given, fixed area. Furthermore, the perturbations are regularised by choosing them within family with a minimum guaranteed level of smoothness. In this way, the optimisation is carried over the perturbation effect only, without having to balance several energy terms as done in [6]. Lastly, by sweeping the area parameter, we can study the perturbation's effect w.r.t. its size.

The second contribution is technical and is to provide a concrete algorithm to calculate the extremal perturbations. First, in the optimisation we must *constrain* the perturbation size to be equal to a target value. To this end, we introduce a new ranking-based *area loss* that can enforce these type of constraints in a stable and efficient manner. This loss, which we believe can be beneficial beyond our perturbation analysis, can be interpreted as a hard constraint, similar to a logarithmic barrier, differing from the soft penalty on the area in Fong and Vedaldi [6]. Second, we construct a parametric family of perturbations with a minimum guarantee amount of smoothness. For this, we use the *(smooth)-max-convolution operator* and a *perturbation pyramid*.

As a final contribution, we extend the framework of perturbation analysis to the intermediate activations of a deep neural network rather than its input. This allows us to explore how perturbations can be used beyond spatial, input-level attribution, to channel, intermediate-layer attribution. When combined with existing visualization techniques such as feature inversion [13, 18, 15, 27], we demonstrate how intermediate-layer perturbations can help us understand which channels are salient for classification.

2. Related work

Backpropagation-based methods. Many attribution techniques leverage backpropagation to track information from the network's output back to its input, or an intermediate layer. Since they are based on simple modifications of the backpropagation algorithm, they only require a single forward and backward pass through the model, and are thus efficient. [22]'s gradient method, which uses unmodified backprop, visualizes the derivative of the network's output w.r.t. the input image. Other works (e.g., DeCovNet [30],

Guided Backprop [25], and SmoothGrad [24]) reduce the noise in the gradient signal by tweaking the backprop rules of certain layers. Other methods generate visualizations by either combining gradients, network weights and/or activations at a specific layer (e.g., CAM [32] and Grad-CAM [21]) or further modify the backpropn rules to have a probabilistic or local approximation interpretation (e.g., LRP [2] and Excitation Backprop [31]).

Several papers have shown that some (but not all) backpropagation-based methods produce the same saliency map regardless of the output neuron being analysed [12], or even regardless of network parameters [1]. Thus, such methods may capture average network properties but may not be able to characterise individual outputs or intermediate activations, or in some cases the model parameters.

Perturbation-based methods. Another family of approaches perturbs the inputs to a model and observes resultant changes to the outputs. Occlusion [30] and RISE [19] occlude an image using regular or random occlusions patterns, respectively, and weigh the changes in the output by the occluding patterns. Meaningful perturbations [6] optimize a spatial perturbation mask that maximally affects a model's output. Real-time saliency [4] builds on [6] and learns to predict such a perturbation mask with a second neural network. Other works have leveraged perturbations at the input [23, 29] and intermediate layers [28] to perform weakly and fully supervised localization.

Approximation-based methods. Black-box models can be analyzed by approximating them (locally) with simpler, more interpretable models. The gradient method of [22] and, more explicitly, LIME [20], do so using linear models. Approximations using decision trees or other models are also possible, although less applicable to visual inputs.

Visualizations of intermediate activations. To characterize a filter's behavior, Zeiler and Fergus [30] show dataset examples from the training set that maximally activate that filter. Similarly, activation maximization [22] learns an input image that maximally activates a filter. Feature inversion [13] learns an image that reconstructs a network's intermediate activations while leveraging a natural image prior for visual clarity. Subsequent works tackled the problem of improving the natural image prior for feature inversion and/or activation maximization [27, 18, 15, 17, 16]. Recently, some methods have measured the performance of single [3, 33] and combinations of [10, 7] filter activations on probe tasks like classification and segmentation to identify which filter(s) encode what concepts.

One difficulty in undertaking channel attribution is that, unlike spatial attribution, where a salient image region is naturally interpretable to humans, simply identifying "important channels" is insufficient as they are not naturally interpretable. To address this, we combine the aforementioned visualization techniques with channel attribution.

3. Method

We first summarize the perturbation analysis of [6] and then introduce our extremal perturbations framework.

3.1. Perturbation analysis

Let $\boldsymbol{x} : \Omega \to \mathbb{R}^3$ be a colour image, where $\Omega = \{0, \dots, H-1\} \times \{0, \dots, W-1\}$ is a discrete lattice, and let Φ be a model, such as a convolutional neural network, that maps the image to a scalar output value $\Phi(\boldsymbol{x}) \in \mathbb{R}$. The latter could be an output activation, corresponding to a class prediction score, in a model trained for image classification, or an intermediate activation.

In the following, we investigate which parts of the input $\boldsymbol{x}$ strongly excite the model, causing the response $\Phi(\boldsymbol{x})$ to be large. In particular, we would like to find a *mask* $\boldsymbol{m}$ assigning to each pixel $u \in \Omega$ a value $\boldsymbol{m}(u) \in \{0, 1\}$, where $\boldsymbol{m}(u) = 1$ means that the pixel strongly contributes to the output and $\boldsymbol{m}(u) = 0$ that it does not.

In order to assess the importance of a pixel, we use the mask to induce a local perturbation of the image, denoted $\hat{\boldsymbol{x}} = \boldsymbol{m} \otimes \boldsymbol{x}$. The details of the perturbation model are discussed below, but for now it suffices to say that pixels for which $\boldsymbol{m}(u) = 1$ are preserved, whereas the others are blurred away. The goal is then to find a small subset of pixels that, when preserved, are sufficient to retain a large value of the output $\Phi(\boldsymbol{m} \otimes \boldsymbol{x})$.

Fong and Vedaldi [6] propose to identify such salient pixels by solving an optimization problem of the type:

$$\boldsymbol{m}_{\lambda,\beta} = \operatorname*{argmax}_{\boldsymbol{m}} \Phi(\boldsymbol{m} \otimes \boldsymbol{x}) - \lambda \|\boldsymbol{m}\|_1 - \beta \mathcal{S}(\boldsymbol{m}). \quad (1)$$

The first term encourages the network's response to be large. The second encourages the mask to select a small part of the input image, blurring as many pixels as possible. The third further regularises the smoothness of the mask by penalising irregular shapes.

The problem with this formulation is that the meaning of the trade-off established by optimizing eq. (1) is unclear as the three terms, model response, mask area and mask regularity, are not commensurate. In particular, choosing different λ and β values in eq. (1) will result in different masks without a clear way of comparing them.

3.2. Extremal perturbations

In order to remove the balancing issues with eq. (1), we propose to constrain the area of the mask to a fixed value (as a fraction $a|\Omega|$ of the input image area). Furthermore, we control the smoothness of the mask by choosing it in a fixed set $\mathcal{M}$ of sufficiently smooth functions. Then, we find the mask of that size that maximizes the model's output:

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}:\ \|\boldsymbol{m}\|_1 = a|\Omega|,\ \boldsymbol{m} \in \mathcal{M}} \Phi(\boldsymbol{m} \otimes \boldsymbol{x}). \quad (2)$$

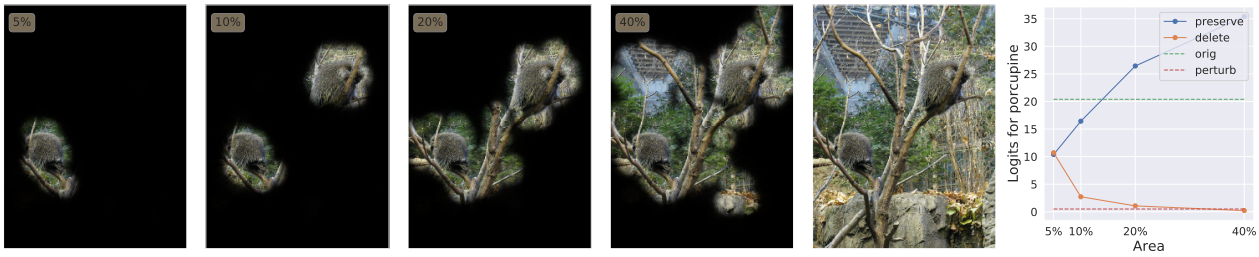


Figure 3: **Extremal perturbations and monotonic effects.** Left: "porcupine" masks computed for several areas a (a in box). Right: $\Phi(\boldsymbol{m}_a \otimes \boldsymbol{x})$ (preservation; blue) and $\Phi((1-\boldsymbol{m}_a)\otimes\boldsymbol{x})$ (deletion; orange) plotted as a function of a. At $a \approx 15\%$ the preserved region scores *higher* than preserving the entire image (green). At $a \approx 20\%$, perturbing the complementary region scores *similarly* to fully perturbing the entire image (red).

Note that the resulting mask is a function of the chosen area a only. With this, we can define the concept of *extremal perturbation* as follows. Consider a lower bound Φ_0 on the model's output (for example we may set $\Phi_0 = \tau\Phi(\boldsymbol{x})$ to be a fraction τ of the model's output on the unperturbed image). Then, we search for the *smallest mask* that achieves at least this output level. This amounts to sweeping the area parameter a in eq. (2) to find

$$a^* = \min\{a : \Phi(\boldsymbol{m}_a \otimes \boldsymbol{x}) \geq \Phi_0\}. \tag{3}$$

The mask $\boldsymbol{m}_{a^*}$ is extremal because preserving a smaller portion of the input image is not sufficient to excite the network's response above Φ_0. This is illustrated in fig. 3.

Interpretation. An extremal perturbation is a single mask $\boldsymbol{m}_{a^*}$ that results in a large model response, in the sense that $\Phi(\boldsymbol{m}_{a^*} \otimes \boldsymbol{x}) \geq \Phi_0$. However, due to extremality, we *also* know that any smaller mask does not result in an equally large response: $\forall \boldsymbol{m} : \|\boldsymbol{m}\|_1 < \|\boldsymbol{m}_{a^*}\|_1 \Rightarrow \Phi(\boldsymbol{m} \otimes \boldsymbol{x}) < \Phi_0$. Hence, a single extremal mask is informative because it characterises a *whole family* of input perturbations.

This connects extremal perturbations to methods like [20, 6], which explain a network by finding a succinct and interpretable description of its input-output mapping. For example, the gradient [22] and LIME [20] approximate the network locally around an input $\boldsymbol{x}$ using the Taylor expansion $\Phi(\boldsymbol{x}') \approx \langle \nabla\Phi(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x}\rangle + \Phi(\boldsymbol{x})$; their explanation is the gradient $\nabla\Phi(\boldsymbol{x})$ and their perturbations span a neighbourhood of $\boldsymbol{x}$.

Preservation vs deletion. Formulation (2) is analogous to what [6] calls the "preservation game" as the goal is to find a mask that preserves (maximises) the model's response. We also consider their "deletion game" obtaining by optimising $\Phi((1-\boldsymbol{m})\otimes\boldsymbol{x})$ in eq. (2), so that the goal is to suppress the response when looking outside the mask, and the hybrid [4], obtained by optimising $\Phi(\boldsymbol{m}\otimes\boldsymbol{x}) - \Phi((1-\boldsymbol{m})\otimes\boldsymbol{x})$, where the goal is to simultaneously preserve the response inside the mask and suppress it outside

3.3. Area constraint

Enforcing the area constraint in eq. (2) is non-trivial; here, we present an effective approach to do so (other approaches like [9] do not encourage binary masks). First, since we would like to optimize eq. (2) using a gradient-based method, we relax the mask to span the full range $[0, 1]$. Then, a possible approach would be to count how many values $\boldsymbol{m}(u)$ are sufficiently close to the value 1 and penalize masks for which this count deviates from the target value $a|\Omega|$. However, this approach requires soft-counting, with a corresponding tunable parameter for binning.

In order to avoid such difficulties, we propose instead to *vectorize and sort* in non-decreasing order the values of the mask $\boldsymbol{m}$, resulting in a vector $\operatorname{vecsort}(\boldsymbol{m}) \in [0,1]^{|\Omega|}$. If the mask $\boldsymbol{m}$ satisfies the area constraint exactly, then the output of $\operatorname{vecsort}(\boldsymbol{m})$ is a vector $\mathbf{r}_a \in [0,1]^{|\Omega|}$ consisting of $(1-a)|\Omega|$ zeros followed by $a|\Omega|$ ones. This is captured by the regularization term: $R_a(\boldsymbol{m}) = \|\operatorname{vecsort}(\boldsymbol{m}) - \mathbf{r}_a\|^2$. We can then rewrite eq. (2) as

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}\in\mathcal{M}} \Phi(\boldsymbol{m}\otimes\boldsymbol{x}) - \lambda R_a(\boldsymbol{m}). \tag{4}$$

Note that we have reintroduced a weighting factor λ in the formulation, so on a glance we have lost the advantage of formulation (2) over the one of eq. (1). In fact, this is not the case: during optimization we simply set λ to be as large as numerics allow it as we expect the area constraint to be (nearly) exactly satisfied; similarly to a logarithmic barrier, λ then has little effect on which mask $\boldsymbol{m}_a$ is found.

3.4. Perturbation operator

In this section we define the perturbation operator $\boldsymbol{m} \otimes \boldsymbol{x}$. To do so, consider a *local perturbation operator* $\pi(\boldsymbol{x}; u, \sigma) \in \mathbb{R}^3$ that applies a perturbation of intensity $\sigma \geq 0$ to pixel $u \in \Omega$. We assume that the lowest intensity $\sigma = 0$ corresponds to no perturbation, i.e. $\pi(\boldsymbol{x}; u, 0) =$

$\boldsymbol{x}(u)$. Here we use as perturbations the Gaussian blur[1]

$$\pi_g(\boldsymbol{x}; u, \sigma) = \frac{\sum_{v\in\Omega} g_\sigma(u-v)\boldsymbol{x}(v)}{\sum_{v\in\Omega} g_\sigma(u-v)}, \quad g_\sigma(u) = e^{-\frac{\|u\|^2}{2\sigma^2}}.$$

The mask $\boldsymbol{m}$ then doses the perturbation spatially: $(\boldsymbol{m}\otimes\boldsymbol{x})(u) = \pi(\boldsymbol{x}; u, \sigma_{\max}\cdot(1-\boldsymbol{m}(u)))$ where $\sigma_{\max}$ is the maximum perturbation intensity.[2]

3.5. Smooth masks

Next, we define the space of smooth masks $\mathcal{M}$. For this, we consider an auxiliary mask $\bar{\boldsymbol{m}} : \Omega \to [0,1]$. Given that the range of $\bar{\boldsymbol{m}}$ is bounded, we can obtain a smooth mask $\boldsymbol{m}$ by convolving $\bar{\boldsymbol{m}}$ by a Gaussian or similar kernel $\boldsymbol{k} : \Omega \to \mathbb{R}_+$[3] via the typical *convolution operator*:

$$\boldsymbol{m}(u) = Z^{-1}\sum_{v\in\Omega} \boldsymbol{k}(u-v)\bar{\boldsymbol{m}}(v) \tag{5}$$

where Z normalizes the kernel to sum to one. However, this has the issue that setting $\bar{\boldsymbol{m}}(u) = 1$ does not necessarily result in $\boldsymbol{m}(u) = 1$ after filtering, and we would like our final mask to be (close to) binary.

To address this issue, we consider the *max-convolution operator*:

$$\boldsymbol{m}(u) = \max_{v\in\Omega} \boldsymbol{k}(u-v)\bar{\boldsymbol{m}}(v). \tag{6}$$

This solves the issue above while at the same time guaranteeing that the smoothed mask does not change faster than the smoothing kernel, as shown in the following lemma (proof in supp. mat.).

Lemma 1. *Consider functions $\bar{\boldsymbol{m}}, \boldsymbol{k} : \Omega \to [0,1]$ and let $\boldsymbol{m}$ be defined as in eq. (6). If $\boldsymbol{k}(0) = 1$, then $\bar{\boldsymbol{m}}(u) \le \boldsymbol{m}(u) \le 1$ for all $u \in \Omega$; in particular, if $\bar{\boldsymbol{m}}(u) = 1$, then $\boldsymbol{m}(u) = 1$ as well. Furthermore, if $\boldsymbol{k}$ is Lipschitz continuous with constant K, then $\boldsymbol{m}$ is also Lipschitz continuous with a constant at most as large as K.*

The $\max$ operator in eq. (6) yields sparse gradients. Thus, to facilitate optimization, we introduce the *smooth max* operator[4], smax, to replace the $\max$ operator. For a function $f(u), u \in \Omega$ and temperature $T > 0$:

$$\operatorname*{smax}_{u\in\Omega;T} f(u) = \frac{\sum_{u\in\Omega} f(u)\exp f(u)/T}{\sum_{u\in\Omega}\exp f(u)/T} \tag{7}$$

[1] Another choice is the fade-to-black perturbation which, for $0 \le \sigma \le 1$, is given by $\pi_f(\boldsymbol{x}; u, \sigma) = (1-\sigma)\cdot\boldsymbol{x}(u)$.

[2] For efficiency, this is implemented by generating a *perturbation pyramid* $\pi(\boldsymbol{x}; \cdot, \sigma_{\max}\cdot l/L)$, $l = 0, \dots, L$ that contains $L+1$ progressively more perturbed versions of the image. Then $\boldsymbol{m}\otimes\boldsymbol{x}$ can be computed via bilinear interpolation by using $(u, \boldsymbol{m}(u))$ as an indices in the pyramid.

[3] It is easy to show that in this case the derivative of the smoothed mask $\||\nabla(\boldsymbol{k} * \bar{\boldsymbol{m}})\|\| \le \|\nabla\boldsymbol{k}\|$ is always less than the one of the kernel.

[4] Not to be confused with the softmax with temperature, as in [8].

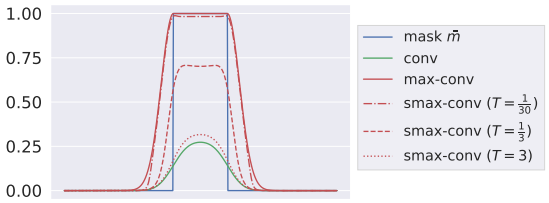


Figure 4: **Convolution operators for smooth masks.** Gaussian smoothing a mask (blue) with the typical convolution operator yields a dampened, smooth mask (green). Our max-convolution operator mitigates this effect while still smoothing (red solid). Our smax operator, which yields more distributed gradients than $\max$, varies between the other two convolution operators (red dotted).

The smax operator smoothly varies from behaving like the mean operator in eq. (5) as $T \to \infty$ to behaving like the $\max$ operator as $T \to 0$ (see fig. 4). This operator is used instead of $\max$ in eq. (6).

Implementation details. In practice, we use a smaller parameterization mask $\bar{\boldsymbol{m}}$ defined on a lattice $\bar{\Omega} = \{0,\dots,\bar{H}-1\}\times\{0,\dots,\bar{W}-1\}$, where the full-resolution mask $\boldsymbol{m}$ has dimensions $H = \rho\bar{H}$ and $W = \rho\bar{W}$. We then modify (6) to perform upsampling in the same way as the standard convolution transpose operator.

4. Experiments

Implementation details. Unless otherwise noted, all visualizations use the ImageNet validation set, the VGG16 network and the preservation formulation (Sec. 3.2). Specifically, $\Phi(\boldsymbol{x})$ is the classification score (before softmax) that network associates to the ground-truth class in the image. Masks are computed for areas $a \in \{0.05, 0.1, 0.2, 0.4, 0.6, 0.8\}$. To determine the optimal area a^* of the extremal perturbations (3), we set the threshold $\Phi_0 = \Phi(\boldsymbol{x})$ (which is the score on the unperturbed image).

Masks are optimised using SGD, initializing them with all ones (everything preserved). SGD uses momentum 0.9 and 1600 iterations. λ is set to 300 and doubled at 1/3 and 2/3 of the iterations and, in eq. (7), $1/T \approx 20$. Before upsampling, the kernel $\boldsymbol{k}(u) = k(\|u\|)$ is a radial basis function with profile $k(z) = \exp\left(\max\{0, z-1\}^2/4\right)$, chosen so that neighbour disks are centrally flat and then decay smoothly.

4.1. Qualitative comparison

Figure 2 shows a qualitative comparison between our method and others. We see that our criterion of $\Phi_0 = \Phi(x)$ chooses fairly well-localized masks in most cases. Masks tend to cover objects tightly, are sharp, and clearly identify a region of interest in the image. Figure 5 shows what the network considered to be most discriminative ($a = 5\%$; e.g.,

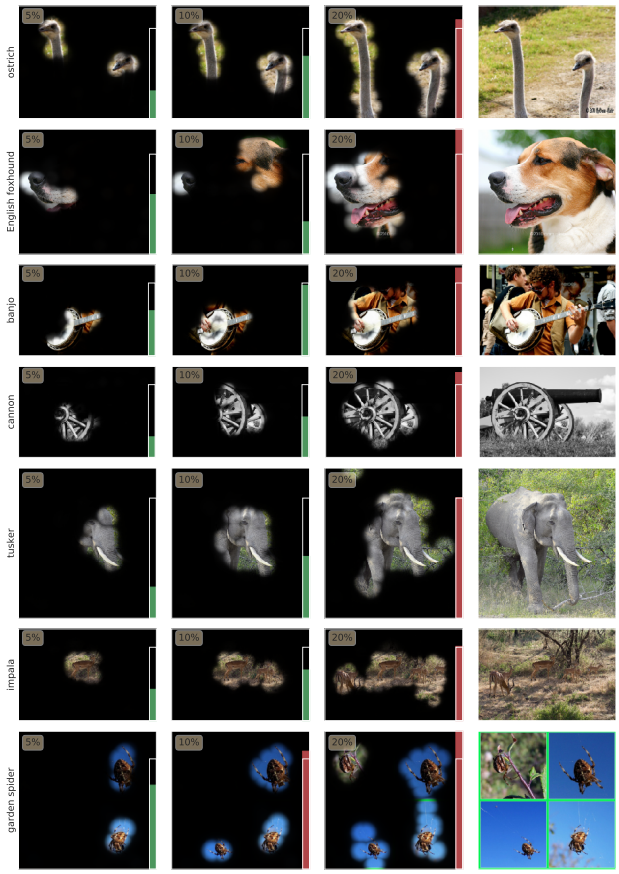


Figure 5: **Area growth.** Although each mask is learned independently, these plots highlight what the network considers to be most discriminative and complete. The bar graph visualizes $\Phi(\boldsymbol{m}_a \odot \boldsymbol{x})$ as a normalized fraction of $\Phi_0 = \Phi(\boldsymbol{x})$ (and saturates after exceeding Φ_0 by 25%).

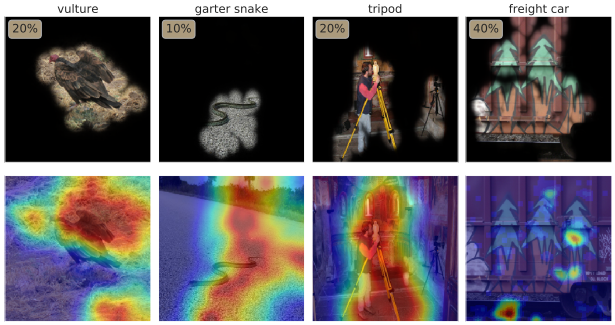


Figure 6: **Comparison with [6].** Our extremal perturbations (top) vs. masks from Fong and Vedaldi [6] (bottom).

banjo fret board, elephant tusk) and complete ($a = 20\%$) as the area increases. We notice that evidence from several objects accumulates monotonically (e.g., impala and spider) and that foreground (e.g., ostrich) or discriminative parts (e.g., dog's nose) are usually sufficient.

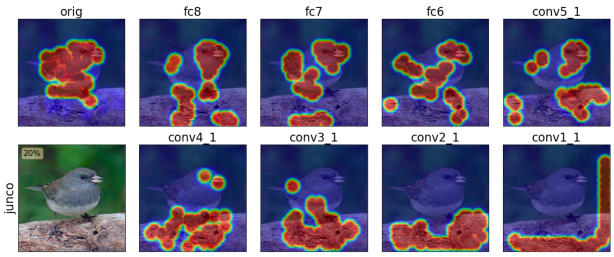


Figure 7: **Sanity check [1].** Model weights are progressively randomized from fc8 to conv1_1 in VGG16, demonstrating our method's sensitivity to model weights.

	VOC07 Test (All/Diff)		*COCO14 Val (All/Diff)*	
Method	*VGG16*	*ResNet50*	*VGG16*	*ResNet50*
Grad	76.3/56.9	72.3/56.8	37.7/31.4	35.0/29.4
DConv	67.5/44.1	68.6/44.7	30.7/23.0	30.0/21.9
Guid.	75.9/53.0	77.2/59.5	39.1/31.4	42.1/35.3
MWP	77.1/56.6	84.4/70.8	39.8/32.8	49.6/43.9
cMWP	79.9/66.5	**90.6**/ 82.2	49.7/44.3	**58.5**/ **53.6**
RISE*	87.3/—	88.9/—	50.7/—	55.6/—
GCAM	86.6/ 74.0	90.4/ **82.3**	**54.2**/ **49.0**	57.3/ 52.3
Ours	**88.7**/ **75.5**	86.3/73.4	53.4/ 47.7	55.7/46.9

Table 1: **Pointing game.** Mean accuracy on the pointing game over the full data splits and a subset of difficult images (defined in [31]). Results from PyTorch re-implementation (* except RISE, which is copied from [19]; RISE VOC results excluded the VOC-defined difficult images).

In fig. 6, we compare our masks to those of Fong and Vedaldi [6]. The stability offered by controlling the area of the perturbation is obvious in these examples. Lastly, we visualize a sanity check proposed in Adebayo et al. [1] in fig. 7 (we use the "hybrid" formulation). Unlike other backprop-based methods, our visualizations become significantly different upon weight randomization (see supp. mat. for more qualitative examples).

4.2. Pointing game

A common approach to evaluate attribution methods is to correlate their output with semantic annotations in images. Here we consider in particular the pointing game of Zhang et al. [31]. For this, an attribution method is used to compute a saliency map for each of the object classes present in the image. One scores a hit if the maximum point in the saliency map is contained within the object; The overall accuracy is the number of hits over number of hits plus misses.

Table 1 shows results for this metric and compares our method against the most relevant work in the literature on PASCAL VOC [5] (using the 2007 test set of 4952 images) and COCO [11] (using the 2014 validation set of $\approx 50k$ im-

ages). We see that our method is competitive with VGG16 and ResNet50 networks. In contrast, Fong and Vedaldi's [6] was not competitive in this benchmark (although they reported results using GoogLeNet).

Implementation details. Since our masks are binary, there is no well defined maximum point. To apply our method to the pointing game, we thus run it for areas $\{0.025, 0.05, 0.1, 0.2\}$ for PASCAL and $\{0.018, 0.025, 0.05, 0.1\}$ for COCO (due to the smaller objects in this dataset). The binary masks are summed and a Gaussian filter with standard deviation equal to 9% of the shorter side of the image applied to the result to convert it to a saliency map. We use the original Caffe models of [31] converted to PyTorch and use the "hybrid" preservation/deletion formulation of our method.

4.3. Monotonicity of visual evidence

Eq. (2) implements the "preservation game" and searches for regions of a given area that *maximally activate* the networks' output. When this output is the confidence score for a class, we hypothesise that hiding evidence from the network would only make the confidence lower, i.e., we would expect the effect of maximal perturbations to be ordered consistently with their size:

$$a_1 \leq a_2 \;\Rightarrow\; \Phi(\boldsymbol{m}_{a_1} \otimes \boldsymbol{x}) \leq \Phi(\boldsymbol{m}_{a_2} \otimes \boldsymbol{x}) \qquad (8)$$

However, this may not always be the case. In order to quantify the frequency of this effect, we test whether eq. (8) holds for all $a_1, a_2 < a^*$, where a^* is the optimal area of the extremal perturbation (eq. (3), where $\Phi_0 = \Phi(\boldsymbol{x})$). Empirically, we found that this holds for 98.45% of ImageNet validation images, which indicates that evidence is in most cases integrated monotonically by the network.

More generally, our perturbations allow us to sort and investigate how information is integrated by the model in order of importance. This is shown in several examples in fig. 5 where, as the area of the mask is progressively increased, different parts of the objects are prioritised.

5. Attribution at intermediate layers

Lastly, we extend extremal perturbations to the *direct* study of the intermediate layers in neural networks. This allows us to highlight a novel use case of our area loss and introduce a new technique for understanding which channels are salient for classification.

As an illustration, we consider in particular channel-wise perturbations. Let $\Phi_l(x) \in \mathbb{R}^{K_l \times H_l \times W_l}$ be the intermediate representation computed by a neural network Φ up to layer l and let $\Phi_{l+} : \mathbb{R}^{K_l \times H_l \times W_l} \to \mathbb{R}$ represent the rest of model, from layer l to the last layer. We then re-formulate the preservation game from eq. (4) as:

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}} \Phi_{l+}(\boldsymbol{m} \otimes \Phi_l(\boldsymbol{x})) - \lambda R_a(\boldsymbol{m}). \qquad (9)$$

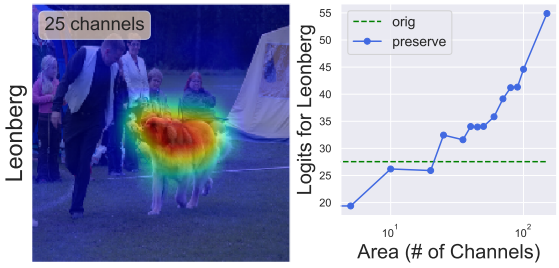


Figure 8: **Attribution at intermediate layers.** Left: This is visualization (eq. (11)) of the optimal channel attribution mask $\boldsymbol{m}_{a^*}$, where $a^* = 25$ channels, as defined in eq. (10). Right: This plot shows that class score monotonically increases as the area (as the number of channels) increases.

Here, the mask $\boldsymbol{m} \in [0,1]^{K_l}$ is a vector with one element per channel which element-wise multiplies with the activations $\Phi_l(\boldsymbol{x})$, broadcasting values along the spatial dimensions. Then, the extremal perturbation $\boldsymbol{m}_{a^*}$ is selected by choosing the optimal area

$$a^* = \min\{a : \Phi_{l+}(\boldsymbol{m}_a \otimes \Phi_l(\boldsymbol{x})) \geq \Phi_0\}. \qquad (10)$$

We assume that the output Φ_{l+} is the pre-softmax score for a certain image class and we set the $\Phi_0 = \Phi(\boldsymbol{x})$ to be the model's predicted value on the unperturbed input (fig. 8).

Implementation details. In these experiments, we use GoogLeNet [26] and focus on layer l =`inception4d`, where $H_l = 14, W_l = 14, K_l = 528$. We optimize eq. (9) for 300 iterations with a learning rate of 10^{-2}. The parameter λ linearly increases from 0 to 1500 during the first 150 iterations, after which $\lambda = 1500$ stays constant. We generate channel-wise perturbation masks for areas $a \in \{1, 5, 10, 20, 25, 35, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 528\}$, where a denotes the number of channels preserved.

The saliency heatmaps in fig. 8 and fig. 9 for channel-wise attribution are generated by summing over the channel dimension the element-wise product of the channel attribution mask and activation tensor at layer l:

$$\boldsymbol{v} = \sum_{k \in K} \boldsymbol{m}^k_{a^*} \otimes \Phi^k_l(\boldsymbol{x}) \qquad (11)$$

5.1. Visualizing per-instance channel attribution

Unlike per-instance input-level spatial attribution, which can be visualized using a heatmap, per-instance intermediate channel attribution is more difficult to visualize because simply identifying important channels is not necessarily human-interpretable. To address this problem, we use feature inversion [14, 18] to find an image that maximises the dot product of the channel attribution vector and the activation tensor (see [18] for more details):

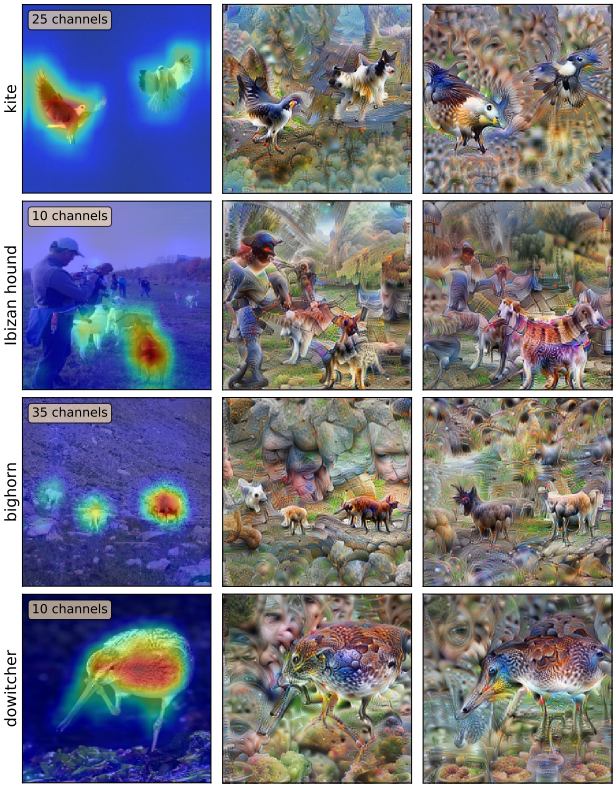


Figure 9: **Per-instance channel attribution visualization.** Left: input image overlaid with channel saliency map (eq. (11)). Middle: feature inversion of original activation tensor. Right: feature inversion of activation tensor perturbed by optimal channel mask $\boldsymbol{m}_{a^*}$. By comparing the difference in feature inversions between un-perturbed (middle) and perturbed activations (right), we can identify the salient features that our method highlights.

$$I^* = \underset{I}{\operatorname{argmax}}\{(\boldsymbol{m}_{a^*} \otimes \Phi_l(\boldsymbol{x})) \cdot \Phi_l(I)\}. \qquad (12)$$

where $\boldsymbol{m}_a^*$ is optimal channel attribution mask at layer l for input image $\boldsymbol{x}$ and $\Phi_l(I)$ is the activation tensor at layer l for image I, the image we are learning.

This inverted image allows us to identify the parts of the input image that are salient for a particular image to be correctly classified by a model. We can compare the feature inversions of activation tensors perturbed with channel mask (right column in fig. 9) to the inversions of original, unperturbed activation tensors (middle column) to get a clear idea of the most discriminative features of an image.

Since the masks are roughly binary, multiplying $\boldsymbol{m}_a^*$ with the activation tensor $\Phi_l(\boldsymbol{x})$ in eq. (12) zeroes out the activations of non-salient channels. Thus, the differences in the feature inversions of original and perturbed activations in fig. 9 highlight the image regions that are encoded by the salient channels identified in our attribution masks (i.e., the channels that are not zeroed out in eq. (12)).

Figure 10: **Discovery of salient, class-specific channels.** By analyzing $\bar{\boldsymbol{m}}_c$, the average over all $\boldsymbol{m}_{a^*}$ for class c (see Sec. 5.2), we automatically find salient, class-specific channels like these. First column: channel feature inversions; all others: dataset examples.

5.2. Visualizing per-class channel attribution

We can also use channel attribution to identify important, class-specific channels. In contrast to other methods, which explicitly aim to find class-specific channels and/or directions at a global level [7, 10, 33], we are able to similarly do so "for free" using only our per-instance channel attribution masks. After estimating an optimal masks $\boldsymbol{m}_{a^*}$ for all ImageNet validation images, we then create a per-class attribution mask $\bar{\boldsymbol{m}}_c \in [0,1]^K$ by averaging the optimal masks of all images in a given class c. Then, we can identify the most important channel for a given class as follows: $k_c^* = \max_{k \in K} \bar{\boldsymbol{m}}_c^k$. In fig. 10, we visualize two such channels via feature inversions. Qualitatively, these feature inversions of channels k_c^* are highly class-specific.

6. Conclusion

We have introduced the framework of extremal perturbation analysis, which avoids some of the issues of prior work that use perturbations to analyse neural networks. We have also presented a few technical contributions to compute such extremal perturbation. Among those, the rank-order area constraint can have several other applications in machine learning beyond the computation of extremal perturbations. Lastly, we have extended the perturbations framework to perturbing intermediate activations and used this to explore a number of properties of the representation captured by a model. In particular, we have visualized, likely for the first time, the difference between perturbed and unperturbed activations using a representation inversion technique.

Acknowledgements. We gratefully acknowledge the support of the Open Philanthropy Project (R.F.), the Rhodes Trust (M.P.), and ESPRC EP/L015897/1 (CDT in Autonomous Intelligent Machines and Systems) (M.P). We would also like to thank Jianming Zhang and Samuel Albanie for helpful advice on re-implementing the Pointing Game [31] in PyTorch.

References

[1] Julius Adebayo, Justin Gilmer, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proc. NeurIPS*, 2018. 3, 6

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 1, 3

[3] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proc. CVPR*, 2017. 3

[4] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Proc. NeurIPS*, 2017. 1, 3, 4

[5] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 111(1):98–136, Jan. 2015. 6

[6] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. ICCV*, 2017. 1, 2, 3, 4, 6, 7

[7] Ruth Fong and Andrea Vedaldi. Net2Vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proc. CVPR*, 2018. 3, 8

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv*, 2015. 5

[9] Hoel Kervadec, Jose Dolz, Meng Tang, Eric Granger, Yuri Boykov, and Ismail Ben Ayed. Constrained-cnn losses for weakly supervised segmentation. *Medical image analysis*, 54:88–99, 2019. 4

[10] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proc. ICML*, 2017. 3, 8

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, 2014. 6

[12] Aravindh Mahendran and Andrea Vedaldi. In *Proc. ECCV*. 3

[13] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proc. CVPR*, 2015. 2, 3

[14] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016. 7

[15] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 3(7):e12, 2018. 2, 3

[16] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proc. CVPR*, 2017. 3

[17] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Proc. NeurIPS*, 2016. 3

[18] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. 2, 3, 7

[19] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proc. BMVC*, 2018. 1, 2, 3, 6

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *Proc. KDD*, 2016. 3, 4

[21] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proc. ICCV*, 2017. 1, 2, 3

[22] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. ICLR workshop*, 2014. 1, 2, 3, 4

[23] Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Proc. ICCV*, 2017. 3

[24] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv*, 2017. 3

[25] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 2015. 1, 2, 3

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015. 7

[27] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proc. CVPR*, 2018. 2, 3

[28] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proc. CVPR*, 2017. 3

[29] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. In *Proc. CVPR*, 2017. 3

[30] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014. 1, 2, 3

[31] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018. 1, 3, 6, 7, 8

[32] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proc. CVPR*, 2016. 3

[33] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Revisiting the importance of individual units in cnns via ablation. *arXiv*, 2018. 3, 8

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Understanding Deep Networks via Extremal Perturbations and Smooth Masks
Publication Status	Published
Publication Details	Ruth Fong*, Mandela Patrick*, and Andrea Vedaldi. Understanding Deep Networks via Extremal Perturbations and Smooth Masks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. * denotes equal contributions.

Student Confirmation

Student Name:	RUTH FONG		
Contribution to the Paper	R.F and A.V. came up with the initial idea. R.F. and M.P. ran experiments and generated the figures. R.F., M.P., and A.V. jointly developed the methods into working algorithms and wrote the paper text.		
Signature 		Date	6 April 2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: ANDREA VEDALDI		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

5

Net2Vec: Quantifying and Explaining how Concepts are Encoded by Filters in Deep Neural Networks

The following paper was presented as a *spotlight* presentation at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) at Salt Lake City, Utah, USA in 2018 (Fong et al., 2018b).[1]

[1]Supplementary materials can be found here: `http://ruthcfong.github.io/files/net2vec_supps.pdf`

Net2Vec: Quantifying and Explaining how Concepts are Encoded by Filters in Deep Neural Networks

Ruth Fong
University of Oxford
ruthfong@robots.ox.ac.uk

Andrea Vedaldi
University of Oxford
vedaldi@robots.ox.ac.uk

Abstract

In an effort to understand the meaning of the intermediate representations captured by deep networks, recent papers have tried to associate specific semantic concepts to individual neural network filter responses, where interesting correlations are often found, largely by focusing on extremal filter responses. In this paper, we show that this approach can favor easy-to-interpret cases that are not necessarily representative of the average behavior of a representation.

A more realistic but harder-to-study hypothesis is that semantic representations are distributed, and thus filters must be studied in conjunction. In order to investigate this idea while enabling systematic visualization and quantification of multiple filter responses, we introduce the Net2Vec framework, in which semantic concepts are mapped to vectorial embeddings based on corresponding filter responses. By studying such embeddings, we are able to show that 1., in most cases, multiple filters are required to code for a concept, that 2., often filters are not concept specific and help encode multiple concepts, and that 3., compared to single filter activations, filter embeddings are able to better characterize the meaning of a representation and its relationship to other concepts.

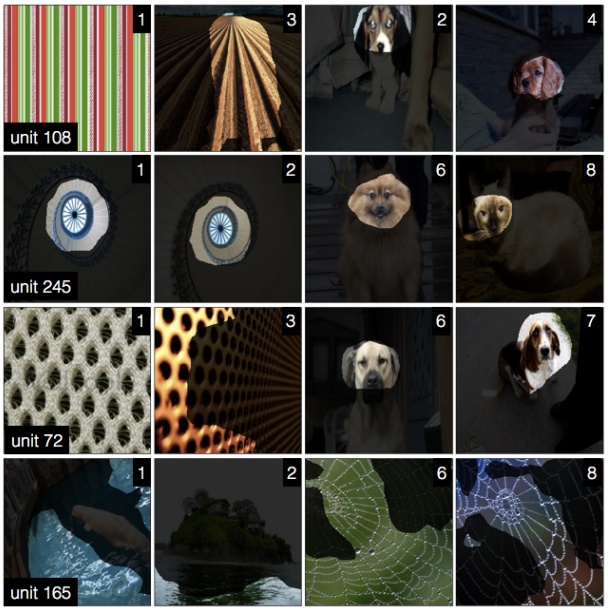


Figure 1. The diversity of BRODEN [4] images that most activate certain AlexNet conv5 filters motivates us to investigate to what extent a single filter encodes a concept fully, without needing other units, and exclusively, without encoding other concepts. An image's corner number n denotes that it is the n-th most maximally activating image for the given filter. Masks were generated by our slightly modified NetDissect [4] approach (section 3.1.1) and are upsampled first before thresholding for smoothness.

1. Introduction

While deep neural networks keep setting new records in almost all problems in computer vision, our understanding of these black-box models remains very limited. Without developing such an understanding, it is difficult to characterize and work around the limitations of deep networks, and improvements may only come from intuition and trial-and-error.

For deep learning to mature, a much better theoretical and empirical understanding of deep networks is thus required. There are several questions that need answering, such as how a deep network is able to solve a problem such as classifying an image, or how it can generalize so well despite having access to limited training data in relation to its own capacity [23]. In this paper, we ask in particular *what a convolutional neural network has learned to do* once training is complete. A neural network can be seen as a sequence of functions, each mapping an input image to some intermediate representation. While the final output of a network is usually easy to interpret (as it provides, hopefully, a solution to the task that the network was trained to solve), the meaning of the intermediate layers is far less clear. Understanding the information carried by these representations is

a first step to understanding how these networks work.

Several authors have researched the possibility that individual filters in a deep network are responsible for capturing particular semantic concepts. The idea is that low-level primitives such as edges and textures are recognized by earlier layers, and more complex objects and scenes by deeper ones. An excellent representative of this line of research is the recent Network Dissection approach by [4]. The authors of this paper introduce a new dataset, BRODEN, which contains pixel-level segmentation for hundreds of low- and high-level visual concepts, from textures to parts and objects. They then study the correlation between extremal filter responses and such concepts, seeking for filters that are strongly responsive for particular ones.

While this and similar studies [24, 22, 10] did find clear correlations between feature responses and various concepts, such an interpretation has intrinsic limitations. This can be seen from a simple counting argument: the number of available feature channels is usually far smaller than the number of different concepts that a neural network may need to encode to interpret a complex visual scene. This suggests that, at the very least, the representation must use combinations of filter responses to represent concepts or, in other words, be at least in part distributed.

Overview. The goal of this paper is to go beyond looking at individual filters, and to study instead what information is captured by **combinations** of neural network filters. In this paper, we conduct a thorough analysis to investigate how semantic concepts, such as objects and their parts, are encoded by CNN filters. In order to make this analysis manageable, we introduce the Net2Vec framework (section 3), which aligns semantic concepts with filter activations. It does so via learned concept embeddings that are used to weight filter activations to perform semantic tasks like segmentation and classification. Our concept vectors can be used to investigate both quantitatively *and* qualitatively the "overlap" of filters and concepts. Our novelty lies in outlining methods that go beyond simply demonstrating that multiple filters better encode concepts that single ones [2, 21] to quantifying and describing how a concept is encoded. Principally, we gain unique, interpretive power by formulating concepts vectors as embeddings.

Using Net2Vec, we look first at two questions (section 4): (1) To what extent are individual filters sufficient to express a concept? Or, are multiple filters required to code for a single concept? (2) To what extent does a filter exclusively code for a single concept? Or, is a filter shared by many, diverse concepts? While answers to these questions depend on the specific filter or concept under consideration, we demonstrate how to **quantify** the "overlap" between filters and concepts and show that there are many cases in which both notions of exclusive overlap do not hold. That is, if we were to interpret semantic concepts and filter activations as corresponding set of images, in the resulting Venn's diagram the sets would intersect partially but neither kind of set would contain or be contained by the other.

While quantifying the relationship between concepts and representation may seem an obvious aim, so far much of the research on explaining how concepts are encoded by deep networks roughly falls into two more qualitative categories: (1) Interpretable visualizations of how single filters encode semantic concepts; (2) Demonstrations of distributive encoding with limited explanatory power of how a concept is encoded. In this work, we present methods that seek to marry the interpretive benefits of single filter visualizations with quantitative demonstrations of how concepts are encoded across multiple filters (section 5).

As part of our analysis, we also highlight the problem with visualizing only the inputs that maximally activate a filter and propose evaluating the power of explanatory visualizations by how well they can explain the whole distribution of filter activations (section 5.1).

2. Related Work

Visualizations. Several methods have been proposed to explain what a single filter encodes by visualizing a real [22] or generated [10, 17, 14] input that most activates a filter; these techniques are often used to argue that single filters substantially encode a concept. In contrast, [20] shows that visualizing the real image patches that most activate a layer's filters after a random basis has been applied also yields semantically, coherent patches. [24, 4] visualize segmentation masks extracted from filter activations for the most confident or maximally activating images; they also evaluate their visualizations using human judgments.

Distributed Encodings. [2] demonstrates that most PASCAL classes require more than a few hidden units to perform classification well. Most similar to [24, 4], [6] concludes that only a few hidden units encode semantic concepts robustly by measuring the overlap between image patches that most activate a hidden unit with ground truth bounding boxes and collecting human judgments on whether such patches encode systematic concepts. [21] compares using individual filter activations with using clusters of activations from all units in a layer and shows that their clusters yielded better parts detectors and qualitatively correlated well with semantic concepts. [3] probes mid-layer filters by training linear classifiers on their activations and analyzing them at different layers and points of training.

3. Net2Vec

With our Net2Vec paradigm, we propose aligning concepts to filters in a CNN by (a) recording filter activations

of a pre-trained network when probed by inputs from a reference, "probe" dataset and (b) learning how to weight the collected probe activations to perform various semantic tasks. In this way, for every concept in the probe dataset, a concept weight is learned for the task of recognizing that concept. The resulting weights can then be interpreted as concept embeddings and analyzed to understand how concepts are encoded. For example, the performance on semantic tasks when using learned concept weights that span all filters in a layer can be compared to when using only a single filter or subset of filters.

In the remainder of the section, we provide details for how we learn concept embeddings by learning to segment (3.1) and classify (3.2) concepts. We also outline how we compare embeddings arising from using only a restricted set of filters, including single filters. Before we do so, we briefly discuss the dataset used to learn concepts.

Data. We build on the BRODEN dataset recently introduced by [4] and use it to primarily probe AlexNet [9] trained on the ImageNet dataset [16] as a representative model for image classification. BRODEN contains over 60,000 images with pixel- and image-level annotations for 1197 concepts across 6 categories: scenes (468), objects (584), parts (234), materials (32), textures (47), and colors (11). We exclude 8 scene concepts for which there were no validation examples. Thus, of the 1189 concepts we consider, all had image-level annotations, but only 682 had segmentation annotations, as only image-level annotations are provided for scene and texture concepts. Note that our paradigm can be generalized to any probe dataset that contains pixel- or image-level annotations for concepts. To compare the effects of different architectures and supervision, we also probe VGG16 [18] conv5_3 and GoogLeNet [19] inception5b trained on ImageNet [16] and Places365 [25] as well as conv5 of the following self-supervised, AlexNet networks: tracking [21], audio [15], objectcentric [5], moving [1], and egomotion [7]. Post-ReLU activations are used.

3.1. Concept Segmentation

In this section, we show how learning to segment concepts can be used to induce concept embeddings using either all the filters available in a CNN layer or just a single filter. We also show how embeddings can be used to quantify the degree of overlap between filter combinations and concepts. This task is performed on all 682 Broden concepts with segmentation annotations, which excludes scene and texture concepts.

3.1.1 Concept Segmentation by a Single Filter

We start by considering single filter segmentation following [4]'s paradigm with three minor modifications, listed below. For every filter k, let a_k be its corresponding activation (at a given pixel location and for a given input image). The $\tau = 0.005$ activation's quantile T_k is determined such that $P(a_k > T_k) = \tau$, and is computed with respect to the distribution $p(a_k)$ of filter activations over all probe images and spatial locations; we use this cut-off point to match [4].

Filter k in layer l is used to generate a segmentation of an image by first thresholding $A_k(\mathbf{x}) > T_k$, where $A_k(\mathbf{x}) \in \mathbb{R}^{H_l \times W_l}$ is the activation map of filter k on input $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ and upsampling the result as needed to match the resolution of the ground truth segmentation mask $L_c(\mathbf{x})$, i.e. $M_k(\mathbf{x}) = S(A_k(\mathbf{x}) > T_k)$, where S denotes a bilinear upsampling function.

Images may contain any number of different concepts, indexed by c. We use the symbol $\mathbf{x} \in X_c$ to denote the probe images that contain concept c. To determine which filter k best segments concept c, we compute a set IoU score. This score is given by the formula

$$\mathrm{IoU}_{\text{set}}(c; M_k, s) = \frac{\sum_{\mathbf{x} \in X_{s,c}} |M_k(\mathbf{x}) \cap L_c(\mathbf{x})|}{\sum_{\mathbf{x} \in X_{s,c}} |M_k(\mathbf{x}) \cup L_c(\mathbf{x})|} \quad (1)$$

which computes the intersection over union (Jakkard index) difference between the binary segmentation masks M_k produced by the filter and the ground-truth segmentation masks L_c. Note that sets are merged for all images in the subset $X_{s,c}$ of the data, where $s \in \{\text{train}, \text{val}\}$. The best filter $k^*(c) = \operatorname{argmax}_k \mathrm{IoU}_{\text{set}}(c; M_k, \text{train})$ is then selected on the training set and the validation score IoU $\mathrm{IoU}_{\text{set}}(c; M_{k^*}, \text{val})$ is reported.

We differ from [4] in the following ways: (1) we threshold before upsampling, in order to more evenly compare to the method described below; (2) we bilinearly upsample without anchoring interpolants at the center of filter receptive fields to speed up the upsampling part of the experimental pipeline; and (3) we determine the best filter for a concept on the training split $X_{\text{train},c}$ rather than X_c whereas [4] does not distinguish a training and validation set.

3.1.2 Concept Segmentation by Filter Combinations

In order to compare single-feature concept embeddings to representations that use filter combinations, we also learn to solve the segmentation task using *combinations* of filters extracted by the neural network. For this, we learn weights $\mathbf{w} \in \mathbb{R}^K$, where K is the number of filters in a layer, to linearly combine thresholded activations. Then, the linear combination is passed through the sigmoid function $\sigma(z) =$

$1/(1+\exp(-z))$ to predict a segmentation mask $M(\mathbf{x};\mathbf{w})$:

$$M(\mathbf{x};\mathbf{w}) = \sigma\left(\sum_k w_k \cdot \mathbb{I}(A_k(\mathbf{x}) > T_k)\right) \quad (2)$$

where $\mathbb{I}(\cdot)$ is the indicator function of an event. The sigmoid is irrelevant for evaluation, for which we threshold the mask predicted by $M(\mathbf{x};\mathbf{w})$ by $\frac{1}{2}$, but has an effect in training the weights $\mathbf{w}$.

Similar to the single filter case, for each concept the weights $\mathbf{w}$ are learned on $X_{\text{train},c}$ and the set IoU score computed on thresholded masks for $X_{\text{val},c}$ is reported. In addition to evaluating on the set IoU score, per-image IoU scores are computed as well:

$$\text{IoU}_{\text{ind}}(\mathbf{x}, c; M) = \frac{|M(\mathbf{x}) \cap L_c(\mathbf{x})|}{|M(\mathbf{x}) \cup L_c(\mathbf{x})|} \quad (3)$$

Note that choosing a single filter is analogous to setting $\mathbf{w}$ to a one-hot vector, where $w_k = 1$ for the selected filter and $w_k = 0$ otherwise, recovering the single-filter segmenter of section 3.1.1, with the output rescaled by the sigmoid function (2).

Training For each concept c, the segmentation concept weights $\mathbf{w}$ are learned using SGD with momentum (lr $= 10^{-4}$, momentum $\gamma = 0.9$, batch size 64, 30 epochs) to minimize a per-pixel binary cross entropy loss weighted by the mean concept size, i.e. 1-α:

$$\mathcal{L}_1 = -\frac{1}{N_{s,c}} \sum_{\mathbf{x} \in X_{s,c}} \alpha M(\mathbf{x};\mathbf{w}) L_c(\mathbf{x}) + (1-\alpha)(1 - M(\mathbf{x};\mathbf{w})(1 - L_c(\mathbf{x})), \quad (4)$$

where $N_{s,c} = |X_{s,c}|$, $s \in \{\text{train}, \text{val}\}$, and $\alpha = 1 - \sum_{\mathbf{x} \in X_{\text{train}}} |L_c(\mathbf{x})|/S$, where $|L_c(\mathbf{x})|$ is the number of foreground pixels for concept c in the ground truth (g.t.) mask for $\mathbf{x}$ and $S = h_s \cdot w_s$ is the number of pixels in g.t. masks.

3.2. Concept Classification

As an alternate task to concept segmentation, the problem of classifying concept (i.e., to tell whether the concept occurs somewhere in the image) can be used to induce concept embeddings. In this case, we discuss first learning embeddings using generic filter combinations (3.2.1) and then reducing those to only use a small subset of filters (3.2.2).

3.2.1 Concept Classification by Filter Combinations

Similar to our segmentation paradigm, for each concept c, a weight vector $\mathbf{w} \in \mathbb{R}^K$ and a bias term $b \in \mathbb{R}$ are learned to combine the spatially-averaged filter activations k; the linear combination is then passed through the sigmoid function σ to obtain the concept posterior probability:

$$f(\mathbf{x};\mathbf{w},b) = \sigma\left(b + \sum_k w_k \cdot \frac{\sum_{i=1}^{H_l}\sum_{j=1}^{W_l} A_{ijk}(\mathbf{x})}{H_l W_l}\right) \quad (5)$$

where H_l and W_l denote the height and width respectively of layer l's activation map $A_k(\mathbf{x})$.

For each concept c, the training images X_{train} are divided into the positive subset $X_{\text{train},c+}$ of images that contain concept c and its complement $X_{\text{train},c-}$ of images that do not. While in general the positive and negative sets are unbalanced, during training, images from the two sets are sampled with equal probability in order to re-balance the data (supp. sec. 1.2). To evaluate performance, we calculate the classification accuracy over a balanced validation set.

3.2.2 Concept Classification by a Subset of Filters

In order to compare using all filters in a layer to just a subset of filters, or even individual filters, we must learn corresponding concept classifiers. Following [2], for each concept c, after learning weights $\mathbf{w}$ as explained before, we choose the top F by their absolute weight $|w_k|$. Then, we learn new weights $\mathbf{w}' \in \mathbb{R}^F$ and bias b' that are used to weight activations from only these F filters. With respect to eq. (5), this is analogous to learning new weights $\mathbf{w}' \in \mathbb{R}^K$, where $w'_k = 0$ for all filters k that are not the top F ones. We train such classifiers for $F \in \{1, 2, 3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 80, 100, 128\}$ for the last three AlexNet layers and for all its layers for the special case $F = 1$, corresponding to a single filter. For comparison, we use this same method to select subsets of filters for the segmentation task on the last layer using $F \in \{1, 2, 4, 8, 16, 32, 64, 128, 160, 192, 224\}$.

4. Quantifying the Filter-Concept Overlap

4.1. Are Filters Sufficient Statistics for Concepts?

We start by investigating a popular hypothesis: whether concepts are well represented by the activation of individual filters or not. In order to quantify this, we consider how our learned weights, which combine information from all filter activations in a layer, compare to a single filter when being used to perform segmentation and classification on BRODEN.

Figure 2 shows that, on average, using learned weights to combine filters outperforms using a single filter on both the segmentation and classification tasks (sections 3.1.1 and 3.2.2) when being evaluated on validation data. The improvements can be quite dramatic for some concepts and starts in conv1. For instance, even for simple concepts like colors, filter combinations outperform individual filters by up to $4\times$ (see supp. figs. 2-4 for graphs on the performance of individual concepts). This suggests that, even if

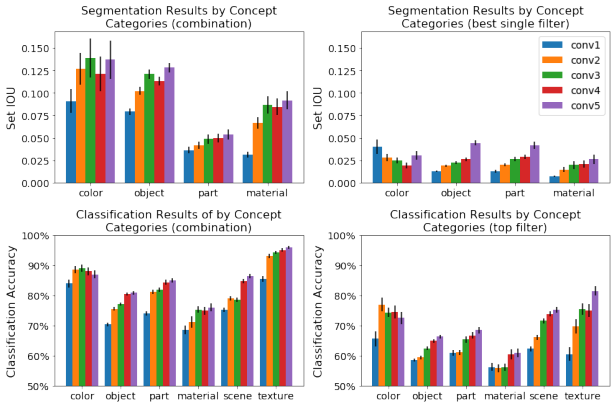


Figure 2. Results by concept category on the segmentation (top) and classification (bottom) tasks show that, on average, using learned weights to combine filters (left) out performs using a single filter (right). Standard error is shown.

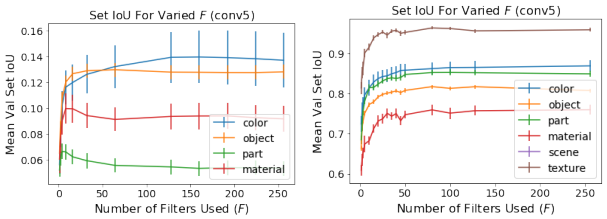


Figure 3. Results by concept category and number of top conv5 filters used for segmentation and classification show that different categories and tasks saturate in performance at different F.

Table 1. Percent of concepts for which the evaluation metric (set IoU for segmentation and accuracy for classification) is equal to or better when using learned weights than the best single filter.

	conv1	**conv2**	**conv3**	**conv4**	**conv5**
Segmentation	91.6%	86.8%	84.0%	82.3%	75.7%
Classification	87.8%	90.2%	85.0%	87.9%	88.1%

filters specific to a concept can be found, these do not optimally encode or fully "overlap" with the concept. In line with the accepted notion that deep layers improve representational quality, task performance generally improves as the layer depth increases, with trends for the color concepts being the notable exception. Furthermore, the average performance varies significantly by concept category and consistently in both the single- and multi-filter classification plots (bottom). This suggests that certain concepts are less well-aligned via linear combination to the filter space.

How many filters are required to encode a concept? To answer this question, we observe how varying the number of top conv5 filters, F, from which we learn concept weights affects performance (section 3.2.2). Figure 3 shows that mean performance saturates at different F for the various concept categories and tasks. For the classification task (right), most concept categories saturate by $F = 50$; however, scenes reaches near optimal performance around $F = 15$, which is much more quickly than that of materials. For the segmentation task (left), performance peaks much earlier at $F = 8$ for materials and parts, $F = 16$ for objects, and $F = 128$ for colors. We also observe performance drops after reaching optimal peaks for materials and parts in the segmentation class. This highlights that the segmentation task is challenging for those concept categories in particular (i.e., object parts are much smaller and harder to segment, materials are most different from network's original ImageNet training examples of objects); with more filters to optimize for, learning is more unstable and more likely to reach a sub-optimal solution.

Failure Cases. While on average our multi-filter approach significantly outperforms a single-filter approach on both segmentation and classification tasks (fig. 2), Table 1 shows that for around 10% of concepts, this does not hold. For segmentation, this percentage increases with layer depth. Upon investigation, we discovered that the concepts for which our learned weights do not outperform the best filter either have very few examples for that concept, i.e. mostly $|X_{\text{train},c}| \in [10, 100]$ which leads to overfitting; or are very small objects, of average size less than 1% of an image, and thus training with the size weighted (4) loss is unstable and difficult, particularly at later layers where there is low spatial resolution. A similar analysis on the classification results shows that small concept dataset size is also causing overfitting in failure cases: Of the 133 conv5 failure cases, 103 had at most 20 positive training examples and all but one had less than 100 positive training examples (supplementary material figs. 7 and 8).

4.2. Are Filters Shared between Concepts?

Next, we investigate the extent to which a single filter is used to encode many concepts. Note that Figure 1 suggests that a single filter might be activated by different concepts; often, the different concepts a filter appears to be activated by are related by a latent concept that may or may not be human-interpretable, i.e., an 'animal torso' filter which also is involved in characterizing animals like 'sheep', 'cow', and 'horse' (fig. 4, supp. fig. 9).

Using the single best filters identified in both the segmentation and classification tasks, we explore how often a filter is selected as the best filter to encode a concept. Figure 5 shows the distribution of how many filters (y-axis) encode how many concepts (x-axis). Interestingly, around 15% of conv1 filters (as well as several in all the other layers) were selected for encoding at least 20 and 30 concepts (# of concepts / # of conv1 filters = 10.7 and 18.6; supp.

Figure 4. AlexNet conv5 filter 66 appears selective for pastoral animal's torso. Validation examples for 'sheep', 'horse', and 'cow' with the highest individual IOU scores are given (masks are upsampled before thresholding for visual smoothness).

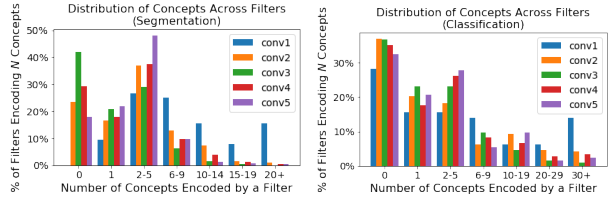


Figure 5. For each filter in a layer, the number of concepts for which it is selected as the best filter in the segmentation (left) and classification (right) tasks is counted and binned.

tbl. 1) for the segmentation and classification tasks respectively and a substantial portion of filters in each layer (except conv1 for the segmentation task) are never selected. The filters selected to encode numerous concepts are not exclusively "overlapped" by a single concept. The filters that were not selected to encode any concepts are likely not be involved in detecting highly discriminative features.

4.3. More Architectures, Datasets, and Tasks

Figure 6 shows segmentation (top) and classification (bottom) results when using AlexNet (AN) conv5, VGG16 (VGG) conv5_3, and GoogLeNet (GN) inception5b trained on both ImageNet (IN) and Places365 (P) as well as conv5 of these self-supervised (SS), AlexNet networks: tracking, audio, objectcentric, moving, and egomotion. GN performed worse than VGG because of its lower spatial resolution (7×7 vs. 14×14); GN-IN inception4e (14×14) outperforms VGG-IN conv5_3 (supp. fig. 11). In [4], GN detects scenes well, which we exclude due to lack of segmentation data. SS performance improves more than supervised networks (5-6x vs. 2-4x), suggesting that SS networks encode

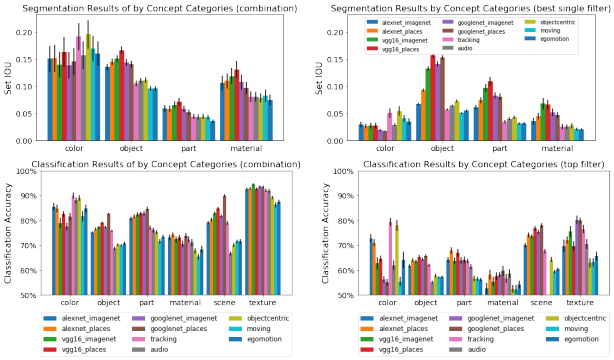


Figure 6. Segmentation (top) and classification (bottom) results for additional networks & datasets.

BRODEN concepts more distributedly.

5. Interpretability

In this section, we propose a new standard for visualizing non-extreme examples, show how the single- and multi-filter perspectives can be unified, and demonstrate how viewing concept weights as embeddings in filter space give us novel explanatory power.

5.1. Visualizing Non-Maximal Examples

Many visual explanation methods demonstrate their value by showing visualizations of inputs that maximally activate a filter, whether that be real, maximally-activating image patches [22]; learned, generated maximally-activated inputs [11, 14]; or filter segmentation masks for maximally-activating images from a probe dataset [4].

While useful, these approaches fail to consider how visualizations differ across the distribution of examples. Figure 7 shows that using a single filter to segment concepts [4] yields IoU_{ind} scores of 0 for many examples; such examples are simply not considered by the set IoU metric. This often occurs because no activations survive the τ-thresholding step, which suggests that a single filter does not consistently fire strongly on a given concept.

We argue that a visualization technique should still work on and be informative for non-maximal examples. In Figure 8, we automatically select and visualize examples at each decile of the non-zero portion of the individual IoU distribution (fig. 7) using both learned concept weights and the best filters identified for each of the visualized categories. For 'dog' and 'airplane' visualizations using our weighted combination method, the predicted masks are informative and salient for most of the examples, even the lowest 10th percentile (leftmost column). Ideally, using this decile sampling method, the visualizations should appear salient even for examples from lower deciles. However, for examples using the best single filter (odd rows), the visualizations are not interpretable until higher deciles (rightmost

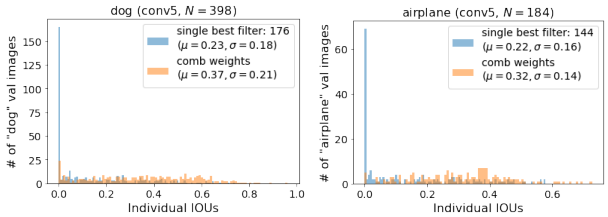


Figure 7. The empirical IoU_{ind} distribution when using the best single filter and the learned weights for 'dog' (left) and 'train' (right) (μ, σ computed on the non-zero part of each distribution).

columns). This is in contrast to the visually appealing, maximally activating examples shown in supp. fig. 13.

5.2. Unifying Single- & Multi-Filter Views

Figure 9 highlights that single filter performance is often strongly, linearly correlated with the learned weights $\mathbf{w}$, thereby showing that individual filter performance is indicative of how weighted it'd be in a linear filter combination. Visually, a filter's set IoU score appears correlated with its associated weight value passed through a ReLU, i.e., $\max(w_k, 0)$. For each of the 682 BRODEN segmentation concepts and each AlexNet layer, we computed the correlation between $\max(\mathbf{w}, 0)$ and $\{\text{IoU}_{\text{set}}(c; M_k, \text{val})\}_{k=1\ldots K}$. By conv3, around 80% of segmentation concepts are significantly correlated ($p < 0.01$): conv1: 47.33%, conv2: 69.12%, conv3: 81.14%, conv4: 79.13%, conv5: 82.47%. Thus, we show how the single filter perspective can be unified with and utilized to explain the distributive perspective: we can quantify how much a single filter k contributes to concept c's encoding from either $\frac{|w_k|}{\|\mathbf{w}\|_1}$ where $\mathbf{w}$ is c's learned weight vector or $\frac{\text{IoUset}(c; M_{k*}, \text{val})}{\text{IoUset}(c; M(\cdot; \mathbf{w}), \text{val})}$.

5.3. Explanatory Power via Concept Embeddings

Finally, the learned weights can be considered as embeddings, where each dimension corresponds to a filter. Then, we can leverage the rich literature [12, 13, 8] on word embeddings derived from textual data to better understand which concepts are similar to each other in network space. To our knowledge, this is the first work that learns semantic embeddings aligned to the filter space of a network from visual data alone. (For this section, concept weights are normalized to be unit length, i.e., $\mathbf{w}' = \frac{\mathbf{w}}{\|\mathbf{w}\|}$).

Table 2 shows the five closest concepts in cosine distance, where 1 denotes that $\mathbf{w}'_1$ is $0°$ from $\mathbf{w}'_2$ and -1 denotes that $\mathbf{w}'_1$ is $180°$ from $\mathbf{w}'_2$. These examples suggest that the embeddings from the segmentation and classification tasks capture slightly different relationships between concepts. Specifically, the nearby concepts in segmentation space appear to be similar-category objects (i.e., animals in the case of 'cat' and 'horse' being nearest to 'dog'), whereas the nearby concepts in classification space appear to be concepts that are related compositionally (i.e., parts of an object in the case of 'muzzle' and 'paw' being nearest to 'dog'). Note that 'street' and 'bedroom' are categorized as scenes and thus lack segmentation annotations.

Understanding the Embedding Space. Table 3 shows that we can also do vector arithmetic by adding and subtracting concept embeddings to get meaningful results. For instance, we observe an analogy relationship between 'grass'−'green' and 'sky'−'blue' and other coherent results, such as non-green, 'ground'-like concepts for 'grass' minus 'green' and floral concepts for 'tree' minus 'wood'. t-SNE visualizations and K-means clustering (see supp. table 2 and supp. figs. 16 and 17) also demonstrate that networks learn meaningful, semantic relationships between concepts.

Comparing Embeddings from Different Learned Representations. The learned embeddings extracted from individual networks can be compared with one another quantitatively (as well as to other semantic representations). Let $d(W) : \mathbb{R}^{C\times K} \to \mathbb{R}^{C\times C} = W \cdot W^T$ compute the cosine distance matrix for C concepts of a given representation (e.g., AlexNet), whose normalized embeddings $\mathbf{w}'$ form the rows of W. Then, $D_{i,j} = \|d(W^i) - d(W^j)\|_2^2$ quantifies the distance between two embedding spaces W^i, W^j, and $D_{i,j,c} = \|d(W^i)_c - d(W^j)_c\|_2^2$ does that for concept c. Figure 10 (left) shows $D_{i,j}$ between 24 embedding spaces: 2 tasks $\times$ 11 network, WordNet (WN), and Word2Vec (W2V) ($C = 501$, the number of BRODEN concepts available for all embeddings; see supp. sec. 3.2.1). It shows that tracking and audio (T, A) classification embeddings are quite different from others, and that classification embeddings (-C) are more aligned to WN and W2V than segmentation ones (-S). Figure 10 (right) shows select mean $D_{i,j,c}$ distances averaged over concept categories. It demonstrates that colors are quite similar between WN and network embeddings and that materials most differ between audio and the WN and W2V embeddings.

6. Conclusion

We present a paradigm for learning concept embeddings that are aligned to a CNN layer's filter space. Not only do we answer the binary questions, "does a single filter encode a concept fully and exclusively?," we also introduce the idea of filter and concept "overlap" and outline methods for answering the scalar extension questions, "to what extent...?" We also propose a more fair standard for visualizing non-extreme examples and show how to explain distributed concept encodings via embeddings. While powerful and interpretable, our approach is limited by its linear nature; future work should explore non-linear ways concepts can be better

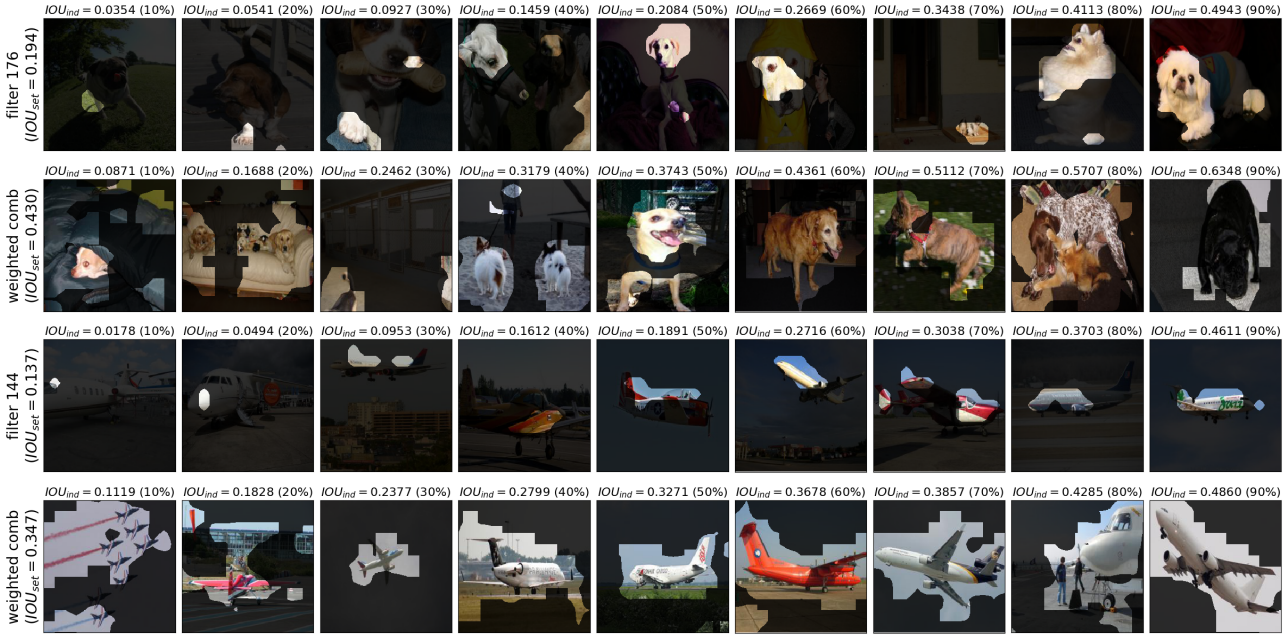


Figure 8. For the 'dog' and 'airplane' concepts, an example is automatically selected at each decile of the non-zero portion of the distribution of individual IoU scores (Figure 7), and the predicted conv5 segmentation masks using the best filter (odd rows) as well as the learned weights (even rows) are overlaid.

Table 2. Nearest concepts (in cos distance) using segmentation (left sub-columns) and classification (right) conv5 embeddings.

dog		**house**		**wheel**			**street**		**bedroom**
cat (0.81)	muzzle (0.73)	building (0.77)	path (0.56)	bicycle (0.86)	headlight (0.66)	n/a	sidewalk (0.74)	n/a	headboard (0.90)
horse (0.73)	paw (0.65)	henhouse (0.62)	dacha (0.54)	motorbike (0.66)	car (0.53)	n/a	streetlight (0.73)	n/a	bed (0.85)
muzzle (0.73)	tail (0.52)	balcony (0.56)	hovel (0.54)	carriage (0.54)	bicycle (0.52)	n/a	license plate (0.73)	n/a	pillow (0.84)
ear (0.72)	nose (0.47)	bandstand (0.54)	chimney (0.53)	wheelchair (0.53)	road (0.51)	n/a	traffic light (0.73)	n/a	footboard (0.82)
tail (0.72)	torso (0.44)	watchtower (0.52)	earth (0.52)	water wheel (0.48)	license plate (0.49)	n/a	windshield (0.71)	n/a	shade (0.74)

Table 3. Vector arithmetic using segmentation, conv5 weights.

grass + blue − green	**grass − green**	**tree − wood**	**person − torso**
sky (0.17)	earth (0.22)	plant (0.36)	foot (0.12)
patio (0.10)	path (0.21)	flower (0.29)	hand (0.10)
greenhouse (0.10)	brown (0.18)	brush (0.29)	grass (0.09)
purple (0.09)	sand (0.16)	bush (0.28)	mountn. pass (0.09)
water (0.09)	patio (0.15)	green (0.25)	backpack (0.09)

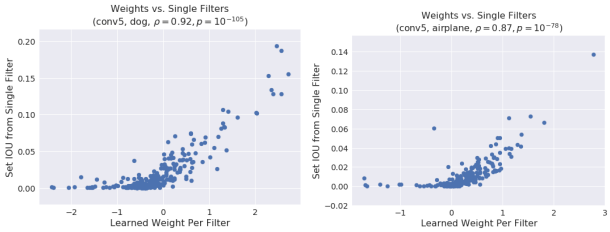


Figure 9. Correlation between learned segmentation weights and each filter's set IoU score for 'dog' (left) and 'airplane' (right).

aligned to the filter space.

Acknowledgements. We gratefully acknowledge the support of the Rhodes Trust for Ruth Fong and ERC 677195-IDIU for Andrea Vedaldi.

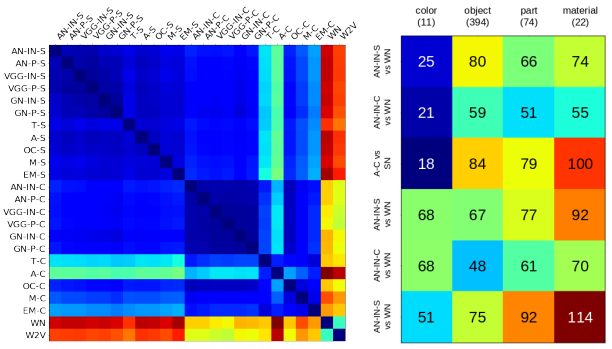


Figure 10. Comparing Net2Vec embeddings quantitatively. Left: Each cell corresponds to distance $D_{i,j}$ for embedding spaces i and j (see section 4.3 for abbreviations). Right: Each cell corresponds to mean distance $D_{i,j,c}$ for each concept category.

References

[1] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *ICCV*, 2015.

[2] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, 2014.

[3] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[4] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017.

[5] R. Gao, D. Jayaraman, and K. Grauman. Object-centric representation learning from unlabeled videos. In *ACCV*, 2016.

[6] A. Gonzalez-Garcia, D. Modolo, and V. Ferrari. Do semantic parts emerge in convolutional neural networks? *IJCV*, 2016.

[7] D. Jayaraman and K. Grauman. Learning image representations tied to ego-motion. In *ICCV*, 2015.

[8] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. In *TACL*, 2014.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[10] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[11] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[13] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *NAACL-HLT*, 2013.

[14] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *NIPS*, 2016.

[15] A. Owens, J. Wu, J. H. McDermott, W. T. Freeman, and A. Torralba. Ambient sound provides supervision for visual learning. In *ECCV*, 2016.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop*, 2014.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[21] J. Wang, Z. Zhang, C. Xie, V. Premachandran, and A. Yuille. Unsupervised learning of object semantic parts from internal states of cnns by population encoding. *arXiv preprint arXiv:1511.06855*, 2015.

[22] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, 2013.

[23] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, 2016.

[24] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

[25] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *T-PAMI*, 2016.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Net2Vec: Quantifying and Explaining How Concepts Are Encoded by Filters in Deep Neural Networks
Publication Status	Published
Publication Details	Ruth Fong and Andrea Vedaldi. Net2Vec: Quantifying and Explaining How Concepts Are Encoded by Filters in Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Student Confirmation

Student Name:	RUTH FONG		
Contribution to the Paper	R.F. proposed the initial idea, developed it into a working algorithm, ran experiments, and generated figures. R.F. and A.V. designed research and wrote the paper text.		
Signature		Date	6 April 2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: ANDREA VEDALDI		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

6

Interactive Similarity Overlays

The following manuscript is being prepared for submission (Fong et al., in prep). Due to its interactive figures, it is best viewed online at the following url (password: "cossim"): `http://ruthcfong.github.io/preview/cossim`.

Interactive Similarity Overlays

An interactive tool for understanding what neural networks consider similar and different.

Hover over different parts of the above images. This interactive visualization shows how similar (or different) a neural network considers different image patches to the current image patch (highlighted in yellow). Try hovering over animal features (e.g., noses, eyes, faces) and background regions.

AUTHORS
Ruth Fong
Alexander Mordvintsev
Andrea Vedaldi
Chris Olah

AFFILIATIONS
University of Oxford
Google Research
University of Oxford
OpenAI

PUBLISHED
Not published yet.

DOI
No DOI yet.

As digital technology has evolved over the past few decades, the ways we interact with it has also evolved. We have moved from typing on a keyboard and viewing a terminal console, to using a mouse and graphical user interface, to employing a variety of touchscreen gestures and voice commands. However, despite the rapid progress in deep learning over the past few years [1, 2, 3, 4], the ways we interact with research artifacts have remained largely unchanged: most visualizations used in research are non-responsive plots, images, and videos.

In this work, we formalize **interactive similarity overlays** — an interactive visualization that highlights how a network "sees" different image patches as similar or different. Our method builds off of prior work on interactive visualizations for understanding CNNs [5, 6, 7, 8, 9, 10] [1] [2] and self-similarity image descriptors [32, 33, 34, 35]. We also demonstrate how similarity overlays can be combined with other visualization techniques, such as non-negative matrix factorization and interactive charts, to more richly explore the learned representations of convolutional neural networks (CNNs). We design our similarity overlays so that they can be easily extended or dropped into the existing workflow of a machine learning researcher. To that end, we release a lightweight package [3] that abstracts away the web development aspects of the visualization; with it, researchers can easily generate similarity overlays for any CNN with a Python interface (e.g., TensorFlow [13], PyTorch [36]).

Interactive similarity overlays allow a user to hover over an image patch and visualize how similar (or different) other image patches are in a CNN representation (see right figure). More precisely, let $s(\mathbf{z_1}, \mathbf{z_2}) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a similarity function and let $f_l(\mathbf{x}) : \mathbb{R}^{3 \times H \times W} \to \mathbb{R}^{D_l \times H_l \times W_l}$ be a function that takes in an input image and returns a 3D tensor (i.e., a CNN up to layer l). [4] Now, let $\mathbf{z} = f_l(\mathbf{x})$ and let $\mathbf{z}_{i,j}$ denote the activation at the (i, j)-th spatial location in $\mathbf{z}$. Then, we visualize the similarity between a given spatial location (i, j) and every other location (u, v), which is given by $s(\mathbf{z}_{i,j}, \mathbf{z}_{u,v})$. This yields a simple and intuitive visualization that allows for easy exploration of different phenomena, which we explore in the rest of this article.

With this technique, we can compare similarities of spatial locations *across* images, as shown in the splash figure above. Within this set of images, we notice that simple background scenes (e.g., those for the dog and cat, flowers, and bird images) are similarly activated despite being visually different. We also observe that a few features, such as eyes, are common across object classes (i.e., different species). Taken together, these observations suggest that CNNs are capable of learning broad and flexible semantic concepts.

Hover over image. The intensity of other patches (lighter denotes more similar) captures the similarity to the highlighted image patch (in yellow).

This multi-image example also highlights the main benefit of interactive similarity overlays, which is their ability to allow users to digest a complex amount of data in an interpretable way. For N images, the full scale of the similarities between all image patches is $\mathcal{O}(N^2 \times H_l^2 \times W_l^2)$. By displaying similarities interactively, we show $\mathcal{O}(N \times H_l \times W_l)$ similarity scores at any given moment, thereby making the data easier to digest.

In the rest of the article, we demonstrate the utility of our interactive similarity overlays in several case studies.

Exploring Different Layers' Representations

First, we consider how interactive similarity overlays help us explore the representations of *different* CNN layers.

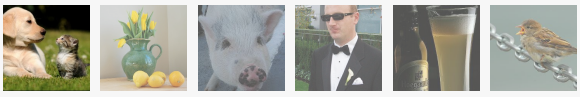
Most prior works have explored layer representations from two perspectives: 1., by exploring how representations in a layer corresponds with different kinds of semantic concepts [38, 39, 40, 41, 42] (e.g., low-level concepts like colors and textures to high-level concepts like objects and scenes), and 2., by visualizing the preferred stimuli of a neuron in specific layers (e.g., activation maximization [43, 44, 45]) or the stimuli that best correspond with a reference activation tensor (e.g., representation inversion [46], caricatures [47]). The former approach typically requires access to manual annotations that define semantic concepts; these are used to "test" a CNN representation and is limited by the breadth and quality of annotation. The latter approach produces a static visualization that often trades off how interpretable a visualization is with how accurately it explains a CNN. [5] In contrast, our interactive visualization similarity does not require manual annotations. Furthermore, thanks to its interactive nature, our visualization accurately renders information about activation similarities in an interpretable interface.

To compare representations of the same input image at different layers, we compute similarity scores *within* each layer and *synchronize* the spatial location being explained across layers (i.e., the

highlighted image patch in yellow). Using this synchronization trick, we first explore the representation of layers with different spatial resolutions. Consistent with some prior work, we find that the earlier layers seem to capture lower-level features like edges while later layers tend to highlight higher-level, semantic features like objects. We also notice that the representations of later layers appear more smooth. [6]

REPRODUCE IN A NOTEBOOK

Layers with different spatial resolutions.



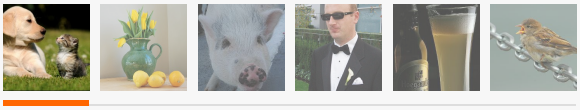
The location of the highlighted image patch (in yellow) has been synchronized across images, such that the overlays show similarity scores with respect to each image's highlighted patch (i.e., no similarity scores were computed between images). Consider exploring edges in mixed3b layers and semantic features (e.g., objects and object parts, like noses and eyes) in mixed4e and mixed5b layers.

We also explore the representation of layers with the same spatial resolution. [7] In this exploration, we confirm our intuition that layer depth affects both the representational smoothness and semantic-

ness.

REPRODUCE IN A NOTEBOOK

Mixed4 Layers.



Notice how the similarity overlays become progressively more smooth and semantic in what they capture as a function of layer depth (i.e., compare similarity overlays across layers for the same spatial location).

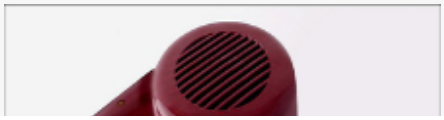
Similarities Across Images

We can also use our visualization to explore representational similarities across images of the same class. One interesting application is to compare correspondences between natural images and generated ones. To that effect, we compute similarity scores across several images, including ones generated to be classified as the same object class [53]. [8] We observe that there seem to be a few correspondences (e.g., awareness of spatial position on an object, such as the handle vs. the nozzle of a blow dryer).

To enhance our visualization and suggest a few corresponding features, we combined our similarity overlays with another visualization tool: matrix factorization. Matrix factorization factors instances into several groups which best explain the variation in a set. In the following example, we use matrix factorization to group activation vectors at different spatial locations (and in different images) into discrete groups. [9] By combining these two visualization techniques together, we glean more information about the CNN representations being visualized than if we were to use either technique alone. Now, we are able to notice and confirm more interesting correspondences (e.g., the correspondence to abstract strokes in a generated image, such as free black strokes corresponding to cords in the blow dryer example).

REPRODUCE IN A CO NOTEBOOK

Comparison with generated images and matrix factors.

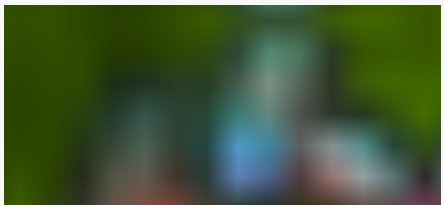


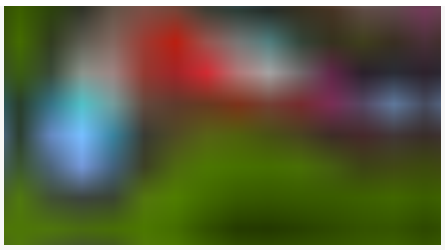



Gillette
supermax
pro
1300





Hover over the large vertical black stroke in the generated image and notice how it appears to correspond to electrical cords in the other images (e.g., the purple component). In one of the images, start hovering over the nozzle opening move to the back of the blow dryer and down the handle. While doing this, notice similar movements reflected in the overlays of other images.

Sensitivity to Geometric Transformations

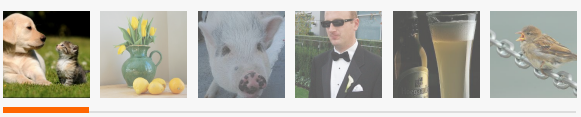
In a final example, we demonstrate how our interactive similarity overlays help us explore how sensitive or invariant a representation is to geometric transformations (e.g., rotation, scale). By systematically transforming an image (e.g., by fixed-degree rotation) and visualizing similarity scores across transformed images, we can visually inspect the impact of a given transformation. We can also combine our overlays with an interactive chart visualization.

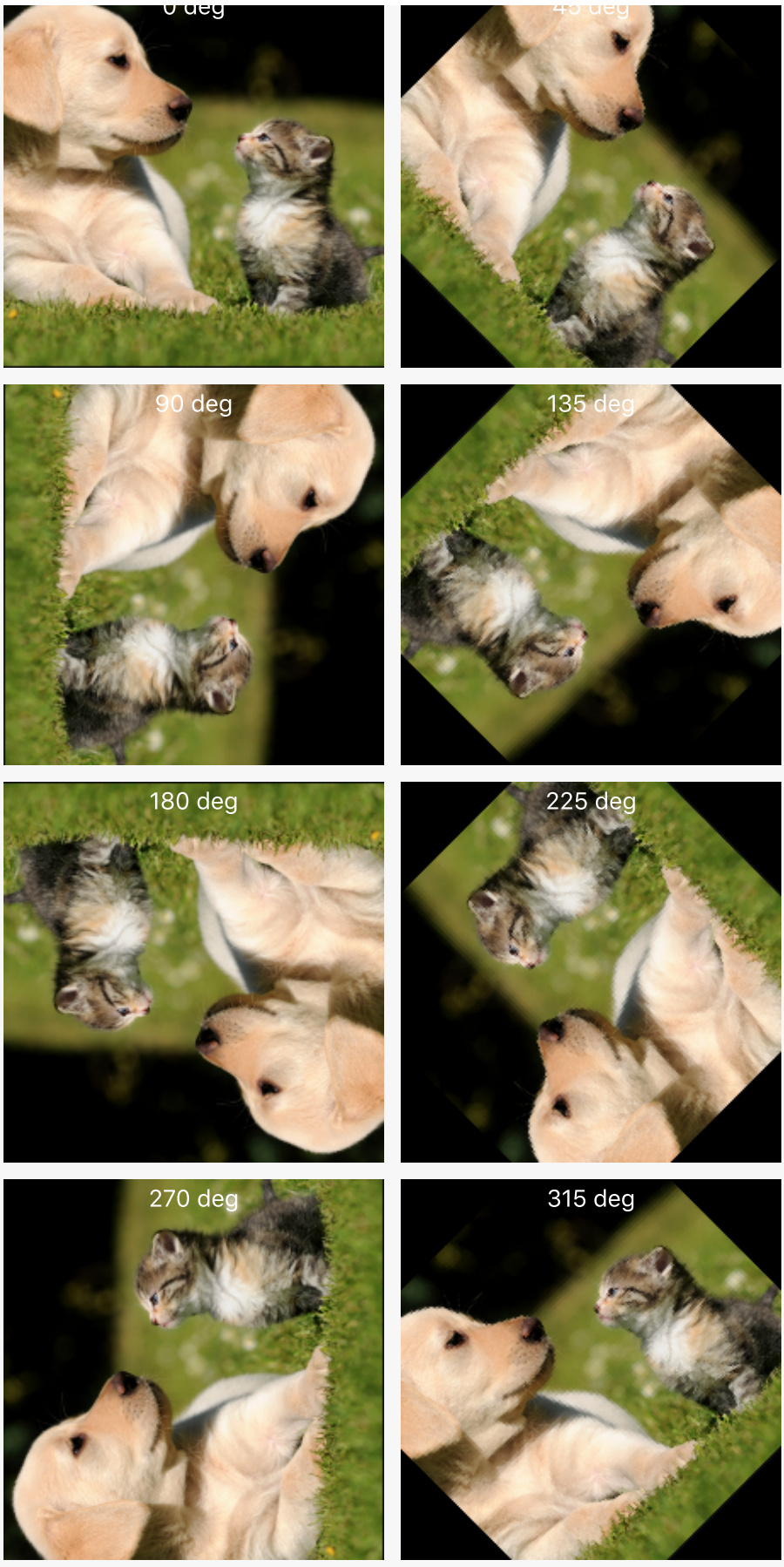
In the rotation example, we show a line chart that displays the similarity scores of the highlighted image patch as well as the corresponding patch in the other transformed images. By leveraging both visualizations, we can quickly notice that more discriminative and oriented features (e.g., animal nose) are more sensitive to rotation than more texture-based, background features (e.g., grass). We also discover rotational sensitivity at image borders; this is likely an artifact from padding the boundaries with zero padding. [10]

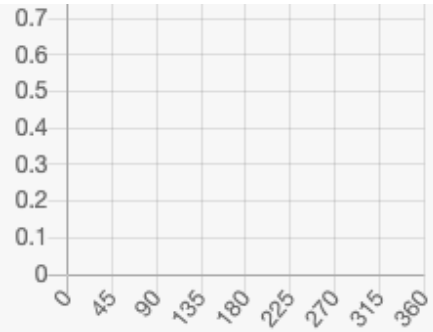
By combining visualizations at different layers of abstraction (e.g., qualitative visualization of similarities across all image patches vs. quantitative visualization of a subset of relevant patches), we demonstrate the utility of combining techniques that operate at different levels of abstraction. [11]

REPRODUCE IN A NOTEBOOK

Rotate.




0 deg
45 deg
90 deg
135 deg
180 deg
225 deg
270 deg
315 deg
1.0
0.9
0.8

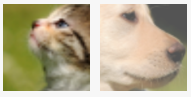


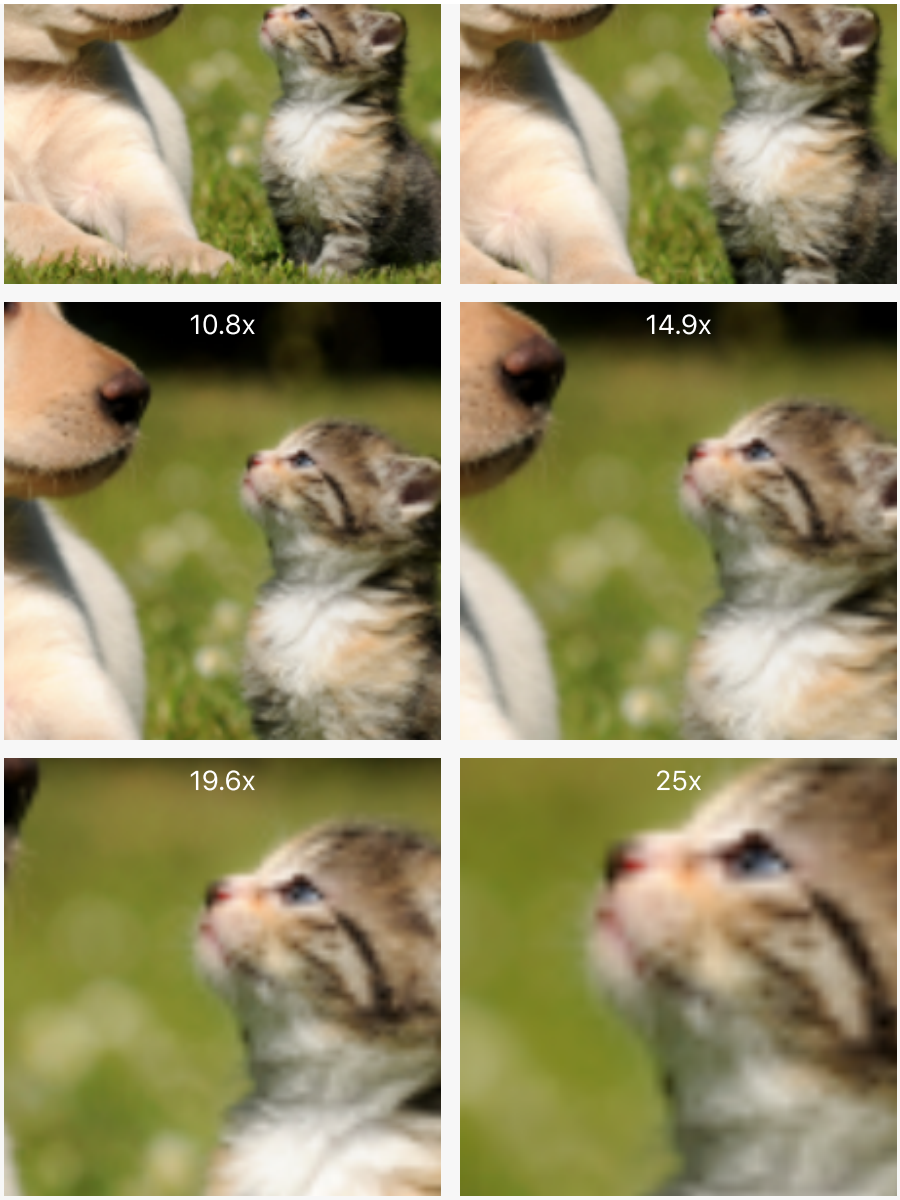
The line chart plots the similarity scores of the hightlighted image patch (e.g., score of 1 for highlighted patch in yellow) and the corresponding patches in the other rotated images. Consider exploring discriminative and oriented features (e.g., animal nose), background features (e.g., grass), and patches along or near the image border.

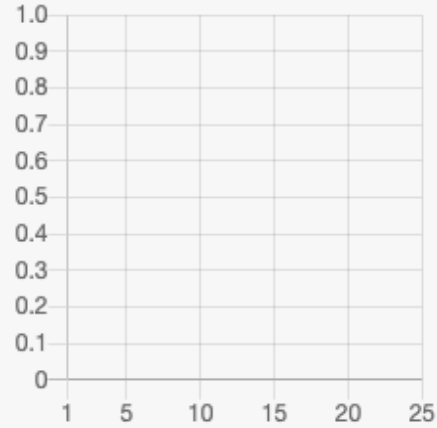
In the scale example, we observe that the spatial relationship of similarities between different features are preserved across scales (e.g., moving a mouse around in one image generates similar "movements" in other images). However, by plotting the similarity scores of the highlighted feature across scales, we see more clearly and quantitatively that similarity scores are somewhat sensitive to large scale changes. This seems to be true for both discriminative features and background ones, though texture-based, background features may be less sensitive (e.g., background grass vs. cat nose).

REPRODUCE IN A CO NOTEBOOK

Scale.







The line chart plots the similarity scores of the highlighted image patch (e.g., score of 1) and the corresponding patches in the other scale-transformed images, if they exist. Consider exploring discriminative features in contrast to background features (e.g., grass vs. cat foreground in 14.9x image).

Conclusion

In summary, we introduce a simple interactive visualization , interactive similarity overlays, which allow a user to investigate the representational similarity of various images. Thanks to its interactive nature, our visualization is both interpretable and faithful to the model being explained. We highlighted how our visualization enables the exploration of a few CNN properties as well as how it can be thoughtfully combined with other techniques to yield further insights.

With a recent movement towards supporting deep learning in Javascript [55, 56] and machine learning research articles with interactive figures [57] , we eagerly expect further work on interactive visualizations for understanding CNNs that can be easily combined with existing tools. [12] To that end, we also release a small package [13] that allows anyone to easily use our interactive similarity overlays without needing to know Javascript. We hope more work is done to empower machine learning practitioners and researchers to easily explore the behavior of their models.

Acknowledgments

We are deeply grateful to the following people for helpful conversations: Tom White, David Bau, Been Kim, Xu Ji, Sam Albanie, Mandela Patrick, Ludwig Schubert, Gabriel Goh, and Nick Cammarata. We are also thankful to the discussion groups organized by Xu Ji within the VGG group and organized by Chris Olah within the Distill Slack workspace. We are also particularly grateful to Tom White for his permission to use his "Perceptual Engines" generated images [53] and to Been Kim for open-sourcing the Gestalt dataset [58] for this project. On the Distill side, we are especially grateful to Ludwig Schubert for his Javascript debugging expertise.

Lastly, this work was made possible by many open source tools, for which we are grateful. In particular, all of our experiments were based on Tensorflow [13] , PyTorch [36] , and Lucid. We built our interactive visualizations using Svelte and Chart.js. We make our results reproducible using Colab notebooks.

Author Contributions

Research: Alex came up with the initial idea of cosine similarity overlays. Ruth developed its applications to interrogate different layers, geometric transformations, etc. Andrea and Chris suggested helpful research directions; in particular, Chris suggested combining similarity overlays with other visualization techniques.

Writing & Diagrams: The text was initially drafted by Ruth and refined by the other authors. The interactive diagrams were designed by all authors. The final notebooks were primarily created by Ruth, based on earlier code and notebooks by Alex and Chris.

Implementation Details

Details

General: Unless otherwise stated, we use the cosine similarity function, $s(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$ as the similarity function with which we compute overlays and visualize GoogLeNet's [37] mixed4d layer as $f_l(\mathbf{x})$.

Non-negative matrix factorization (NNMF): For each object class (e.g., blow dryer), 10 (out of 50) real images from the ImageNet [59] validation set for that class were selected. NNMF was computed on the set of 11 images (10 real images and 1 generated image [53]). By default, 4 components were computed; upon visual inspection, this was reduced to 3 for a few object classes. We found that 10 real images was an ideal number of images to use to compute salient components and chose images were somewhat visual similar to one another (e.g., simple backgrounds for blow dryer example). We then selected 5 of the 10 real images to be shown in the figure.

Footnotes

1. See [11, 12] for a survey of visualizations for machine learning and deep learning respectively. [↩]

2. A number of works use interactivity to navigate a visual interface [13, 14], focus on visualizing a single model or dataset deeply for pedogogical purposes [15, 16, 17, 18, 19, 20, 21, 22], treat a model as a black-box by visualizing model inputs and outputs (not internal components) [23, 24, 14], and/or explore other kinds of models or feature representations (e.g., GANs [25, 21, 26], RNNs [27, 28, 22], etc. [29, 30, 15, 31]). In contrast, our work is an interactive visualization that can be used to explain the internal representation of any CNN model. [↩]

3. github.com/ruthcfong/interactive_overlay [↩]

4. Unless otherwise stated, we use the cosine similarity function, $s(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$, and GoogLeNet's [37] mixed4d layer as $f_l(\mathbf{x})$. We chose the cosine similarity function because it is the noramlized dot product of two vectors, which quantifies the angle between them (i.e., it captures the *directional* similarity of two vectors). [↩]

5. In the case of feature visualizations, interpretability refers to how easily interpretable a visualization is (e.g., some feature visualizations are highly unnatural and are hard to reason about), while fidelity refers to how accurately a visualization explains a given model component (e.g., neuron or activation tensor; some feature visualizations rely on strong priors [48, 49, 50, 51, 52] in order to be more interpretable by trading off fidelity). Refer to [45] for a discussion of this tradeoff. [↩]

6. Neighboring spatial locations possess similar scores in smooth representations. [↩]

7. That is, layers that output tensors of the same spatial dimensions (i.e., H_l and W_l). [↩]

8. These generated images were constrained by brush strokes and inks to appear like modern art. See [53] for more details about their generation process (the prints are available for purchase here). [↩]

9. See "Implementation Details" at the end of the article for more information about how this figure was generated. [↩]

10. Highlight a boundary pixel and move inward towards the center of the image; you will notice that the ripple effect in similarity scores shown in the line chart becomes smaller as your cursor moves towards the image center. Due to large receptive fields, patches in between the image border and center may still be partially affected by boundary effects. [↩]

11. See [54] for further discussion on the benefits of combining multiple layers of abstraction. [↩]

12. Currently, there are a few open-source packages (e.g., TensorFlow's Lucid and PyTorch's Captum) that implement several CNN visualization methods and support combining techniques. [↩]

13. github.com/ruthcfong/interactive_overlay [↩]

References

1. ImageNet Classification with Deep Convolutional Neural Networks [PDF]
   Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS).

2. Mask R-CNN [PDF]
   He, K., Gkioxari, G., Dollar, P. and Girshick, R., 2017. Proceedings of the IEEE International Conference on Computer Vision (ICCV).

3. Mastering the game of Go without human knowledge [link]
   Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and others,, 2017. Nature, Vol 550(7676), pp. 354--359. Nature Publishing Group.

4. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [link]
   Devlin, J., Chang, M., Lee, K. and Toutanova, K., 2018. arXiv preprint arXiv:1810.04805.

5. Understanding Neural Networks Through Deep Visualization [link]
   Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. and Lipson, H., 2015. Deep Learning Workshop, International Conference on Machine Learning (ICML).

6. A Neural Network Playground [link]
   Tensorflow, P., 2017.

7. The Building Blocks of Interpretability [link]
   Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. and Mordvintsev, A., 2018. Distill, Vol 3(3), pp. e10.

8. Exploring Neural Networks with Activation Atlases [link]
   Carter, S., Armstrong, Z., Schubert, L., Johnson, I. and Olah, C., 2019. Distill, Vol 4(3), pp. e15.

9. Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations [link]
   Hohman, F., Park, H., Robinson, C. and Chau, D.H., 2020. IEEE Transactions on Visualization and Computer Graphics (TVCG). IEEE.

10. Understanding and Visualizing Data Iteration in Machine Learning [PDF]
    Hohman, F., Wongsuphasawat, K., Kery, M.B. and Patel, K., 2020. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.

11. What you see is what you can change: Human-centered machine learning by interactive visualization [link]
    Sacha, D., Sedlmair, M., Zhang, L., Lee, J.A., Peltonen, J., Weiskopf, D., North, S.C. and Keim, D.A., 2017. Neurocomputing, Vol 268, pp. 164--175. Elsevier.

12. Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers [PDF]
    Hohman, F., Kahng, M., Pienta, R. and Chau, D.H., 2018. IEEE Transactions on Visualization and Computer Graphics (TVCG). IEEE.

13. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [PDF]
    Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I.J., Harp, A., Irving, G., Isard, M., Jia, Y., J{\'{o}}zefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Man{\'{e}}, D., Monga, R., Moore, S., Murray, D.G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P.A., Vanhoucke, V., Vasudevan, V., Vi{\'{e}}gas, F.B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2016. arXiv preprint arXiv:1603.04467.

14. The What-If Tool: Interactive Probing of Machine Learning Models [link]
    Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viegas, F. and Wilson, J., 2019. IEEE Transactions on Visualization and Computer Graphics (TVCG), Vol 26(1), pp. 56--65. IEEE.

15. How to Use t-SNE Effectively [link]

Wattenberg, M., Viegas, F. and Johnson, I., 2016. Distill, Vol 1(10), pp. e2.

16. Four Experiments in Handwriting with a Neural Network [link]
    Carter, S., Ha, D., Johnson, I. and Olah, C., 2016. Distill, Vol 1(12), pp. e4.

17. ShapeShop: Towards Understanding Deep Learning Representations via Interactive Experimentation [PDF]
    Hohman, F., Hodas, N. and Chau, D.H., 2017. Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems. ACM.

18. DrawNet [link]
    Torralba, A., 2017.

19. Direct-Manipulation Visualization of Deep Networks [PDF]
    Smilkov, D., Carter, S., Sculley, D., Viegas, F.B. and Wattenberg, M., 2017. arXiv preprint arXiv:1708.03788.

20. Adversarial-Playground: A Visualization Suite Showing How Adversarial Examples Fool Deep Learning [PDF]
    Norton, A.P. and Qi, Y., 2017. 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 1--4.

21. GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation [link]
    Kahng, M., Thorat, N., Chau, D.H.P., Viegas, F.B. and Wattenberg, M., 2018. IEEE Transactions on Visualization and Computer Graphics (TVCG), Vol 25(1), pp. 1--11. IEEE.

22. Visualizing memorization in RNNs [link]
    Madsen, A., 2019. Distill, Vol 4(3), pp. e16.

23. Now anyone can explore machine learning, no coding required [link]
    Webster, B., 2017.

24. Facets - Visualizations for ML Datasets [link]
    PAIR, G., 2017.

25. Generative Visual Manipulation on the Natural Image Manifold [link]
    Zhu, J., Krahenbuhl, P., Shechtman, E. and Efros, A.A., 2016. Proceedings of European Conference on Computer Vision (ECCV).

26. Semantic Photo Manipulation with a Generative Image Prior [link]
    Bau, D., Strobelt, H., Peebles, W., Wulff, J., Zhou, B., Zhu, J. and Torralba, A., 2019. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH), Vol 38(4).

27. LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks [link]
    Strobelt, H., Gehrmann, S., Pfister, H. and Rush, A.M., 2017. IEEE Transactions on Visualization and Computer Graphics (TVCG), Vol 24(1), pp. 667--676. IEEE.

28. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models [link]
    Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H. and Rush, A.M., 2018. IEEE Transactions on Visualization and Computer Graphics (TVCG), Vol 25(1), pp. 353--363. IEEE.

29. EnsembleMatrix: Interactive Visualization to Support Machine Learning with Multiple Classifiers [PDF]
    Talbot, J., Lee, B., Kapoor, A. and Tan, D.S., 2009. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1283--1292.

30. Visualizing MNIST: An Exploration of Dimensionality Reduction, 2014 [link]
    Olah, C..

31. Gamut: A Design Probe to Understand How Data Scientists Understand Machine Learning Models [link]
    Hohman, F., Head, A., Caruana, R., DeLine, R. and Drucker, S.M., 2019. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems

Factors in Computing Systems.

32. Matching Local Self-Similarities across Images and Videos [link]
Shechtman, E. and Irani, M., 2007. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

33. Global and Efficient Self-Similarity for Object Classification and Detection [PDF]
Deselaers, T. and Ferrari, V., 2010. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

34. Static and space-time visual saliency detection by self-resemblance [link]
Seo, H.J. and Milanfar, P., 2009. Journal of vision, Vol 9(12), pp. 15--15. The Association for Research in Vision and Ophthalmology.

35. Multiple Kernels for Object Detection [PDF]
Vedaldi, A., Gulshan, V., Varma, M. and Zisserman, A., 2009. Proceedings of the IEEE International Conference on Computer Vision (ICCV).

36. Automatic differentiation in PyTorch [link]
Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A., 2017. Proceedings of the Neural Information Processing Systems (NIPS).

37. Going Deeper with Convolutions [PDF]
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

38. Analyzing the Performance of Multilayer Neural Networks for Object Recognition [link]
Agrawal, P., Girshick, R. and Malik, J., 2014. Proceedings of the European Conference on Computer Vision.

39. Object Detectors Emerge in Deep Scene CNNs [PDF]
Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. and Torralba, A., 2015. Proceedings of the International Conference on Learning Representations (ICLR).

40. Do Semantic Parts Emerge in Convolutional Neural Networks? [link]
Gonzalez-Garcia, A., Modolo, D. and Ferrari, V., 2018. International Journal of Computer Vision (IJCV), Vol 126(5), pp. 476--494. Springer.

41. Network Dissection: Quantifying Interpretability of Deep Visual Representations [link]
Bau, D., Zhou, B., Khosla, A., Oliva, A. and Torralba, A., 2017. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

42. Net2Vec: Quantifying and Explaining How Concepts Are Encoded by Filters in Deep Neural Networks [PDF]
Fong, R. and Vedaldi, A., 2018. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

43. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps [link]
Simonyan, K., Vedaldi, A. and Zisserman, A., 2014. Proceedings of the Workshop at the International Conference on Learning Representations (ICLR).

44. Visualizing and Understanding Convolutional Networks [PDF]
Zeiler, M.D. and Fergus, R., 2014. Proceedings of the European Conference on Computer Vision (ECCV).

45. Feature Visualization [link]
Olah, C., Mordvintsev, A. and Schubert, L., 2017. Distill, Vol 2(11), pp. e7.

46. Understanding Deep Image Representations by Inverting Them [PDF]
Mahendran, A. and Vedaldi, A., 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

47. Inceptionism: Going deeper into neural networks [HTML]
Mordvintsev, A., Olah, C. and Tyka, M., 2015. Google Research Blog. Retrieved June, Vol 20(14), pp. 5.

48. Generating Images with Perceptual Similarity Metrics based on Deep Networks [link]
Dosovitskiy, A. and Brox, T., 2016. Proceedings of the International Conference on Neural Information Processing Systems (NIPS).

49. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks [link]
Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T. and Clune, J., 2016. Proceedings of the International Conference on Neural Information Processing Systems (NIPS).

50. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space [link]
Nguyen, A., Clune, J., Bengio, Y., Dosovitskiy, A. and Yosinski, J., 2017. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

51. Deep Image Prior [link]
Ulyanov, D., Vedaldi, A. and Lempitsky, V.S., 2018. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

52. Differentiable Image Parameterizations [link]
Mordvintsev, A., Pezzotti, N., Schubert, L. and Olah, C., 2018. Distill, Vol 3(7), pp. e12.

53. Shared Visual Abstractions [PDF]
White, T., 2019. Proceedings of the NeurIPS Workshop on Shared Visual Representations in Human and Machine Intelligence (SVRHM).

54. Up and Down the Ladder of Abstraction: A Systematic Approach to Interactive Visualization [link]
Victor, B., 2011.

55. Tensorflow. js: Machine learning for the web and beyond [PDF]
Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D. and others,, 2019. arXiv preprint arXiv:1901.05350.

56. Magenta.js: A JavaScript API for Augmenting Creativity with Deep Learning [PDF]
Roberts, A., Hawthorne, C. and Simon, I., 2018. Joint Workshop on Machine Learning for Music (ICML).

57. Machine Learning Should Be Clear, Dynamic and Vivid. Distill is Here to Help. [link]
.

58. Do neural networks show Gestalt phenomena? An exploration of the law of closure [PDF]
Kim, B., Reif, E., Wattenberg, M. and Bengio, S., 2019. arXiv preprint arXiv:1903.01069.

59. ImageNet Large Scale Visual Recognition Challenge [link]
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and others,, 2015. International journal of computer vision, Vol 115(3), pp. 211--252. Springer.

7 Discussion

Contents

In this section, we discuss the impact of the work presented in the earlier chapters. For each research area, we discuss the impact of our research in relation to other works that build off our work (*i.e.* extensions), are in close dialogue with our research (*i.e.* related works), and are quite similar to ours (*i.e.* independent discoveries). We then outline a few promising future directions that have not yet been explored: some related to the works presented in this thesis as well as a few

under-explored directions in the interpretability research space (section 7.4). Finally, we summarize the main contributions of this thesis in section 7.5.

This chapter can be seen as a continuation of chapter 2, which surveyed related works that generally preceded ours. Together, both chapters serve to illustrate that research rarely happens in a vacuum but is often the product of interacting with, responding to, and being inspired by many other related works. In this chapter, we aim to shed some light on some ongoing discussions (both formally via research papers and informally via conversation) among researchers working on CNN interpretability. We also hope that this chapter highlights that there is much more work to be done in terms of increasing our understanding of CNNs and outlines a few ideas to that effect.

7.1 Attribution heatmaps

7.1.1 Extensions of meaningful perturbations (Fong et al., 2017)

Building off meaningful perturbations (Fong et al., 2017), Dabkowski et al., 2017 train a segmentation network to predict masks similar to those produced by Fong et al., 2017 in order to produce attribution heatmaps in real-time (*a.k.a.* **real-time saliency**). While Dabkowski et al., 2017 indeed yields a significant computational speedup, the use of a trained network limits its ability to produce explanations for instances outside the training domain of the network. Dabkowski et al., 2017 also formalize the notions of the smallest sufficient region (SSR) and the smallest destroying region (SDR), which respectively connote the smallest region that, when shown to an image classifier, is sufficient for correct prediction and the smallest region that, when its component is shown to an image classifier, is sufficient for misclassification. Greydanus et al., 2018 also extend meaningful perturbations to visually explain the policy decisions of deep reinforcement learning agents playing Atari games. Chang et al., 2017 extend Fong et al., 2017 by learning to generate a mask distribution (rather than a single mask) as well as generating the value of replacement pixels (rather than simply blurring them). While mathematically interesting, Chang et al., 2017 do not demonstrate their Variational Dropout Saliency Map (**VDSM**) on complex, real-world datasets; instead, they show results on the MNIST dataset (LeCun et al., 2010). Similarly, Chang et al., 2019 explore learning counterfactual explanations by using a generative model to fill in masked regions instead of blurring them as was done in Fong et al., 2017; they argue that

this encourages the produced explanation to be more realistic and comparable to samples from the training distribution.

7.1.2 Related independent work to extremal perturbations (Fong et al., 2019a)

XRAI (Kapishnikov et al., 2019) independently introduce a perturbation-based saliency technique with a few similar qualities to our extremal perturbations work (Fong et al., 2019a). Like extremal perturbations, XRAI learns *area-constrained* attribution heatmaps. However, while extremal perturbations allows for free-form, smooth heatmaps to be learned, XRAI constrains the space of possible attribution heatmaps to superpixels computed from low-level image features (*e.g.* edges). This constraint is likely imposed in order to limit the potential for XRAI to learn an adversarial mask (see Fong et al., 2017 in chapter 3 for a discussion on adversarial artifacts). Furthermore, while extremal perturbations learns an attribution mask via optimization, XRAI finds one via a *greedy algorithm*[1] that successively adds superpixels to a work-in-progress attribution mask until the area budget is used up. Because of these differences, we argue that our extremal perturbations method is preferable because it learns an attribution heatmap by dynamically considering a wide range of free-form, smooth masks.

7.1.3 Discussion on metrics and attribution method design

In meaningful perturbations (Fong et al., 2017) (as well as in an extended book chapter (Fong et al., 2019b)), we discuss the need for attribution methods and evaluation metrics to be carefully designed to truly understand what regions are responsible for a particular model's decision.

In Fong et al., 2017, we outline several desiderata for explanation-producing systems (and by extension their evaluation metrics), with a particular focus on attribution heatmaps. Specifically, we highlight the need for an explanation to be both *faithful*[2] to and *maximally informative*[3] (*i.e.* meaningful) about the model being explained.

[1] A greedy algorithm is one that selects the best local choice (*i.e.* best option at that time) at each step in the algorithm. Unless proven, greedy algorithms typically do not learn the globally optimal (*i.e.* overall best) solution (Black, 2005). See the Wikipedia article on "Greedy algorithm" for more details.

[2] To be faithful to a model is to provide an accurate explanation of its behavior.

[3] To be maximally informative about a model is to provide the most useful or meaningful information about a certain aspect of its behavior.

To ensure that an attribution heatmap is a faithful explanation, we highlight the superiority of saliency methods that are *testable* or *falsifiable* in Fong et al., 2019b. Our meaningful perturbations and extremal perturbations methods are inherently falsifiable: the produced explanation mask is meant to either maximally destroy or maximally preserve a model's class prediction when used to perturb an image. This explanation can be easily tested as well as compared to other potential regions. A number of propagation-based methods are not testable by design because they typically are based on propagating and visualizing a signal through many CNN layers. Because of this, the meaning of that signal and the highlighted regions from the resultant heatmap is often unclear (*e.g.* what does it mean that a specific region is highlighted?).

This does not imply that that there is not a clear *interpretation* to an attribution method. For example, the gradient method (Simonyan et al., 2014) has a clear interpretation: it visualizes the gradient of a model's output with respect to its input. However, a clear interpretation of a method does not ensure that produced heatmaps are falsifiable or even faithful. For instance, the gradient is not necessarily a faithful explanation of a model's behavior on a specific input.[4]

In contrast, the meaning of highlighted regions in perturbation-based attribution heatmaps — *e.g.* occlusion (Zeiler et al., 2014), RISE (Petsiuk et al., 2018) — is arguably more clear because they are grounded in real-world edits to an image: a highlighted region means that perturbing the input there will affect the model's output prediction in some way. Our meaningful and extremal perturbation masks go a step further and by design highlight the regions that are most critical to a model's output prediction. Furthermore, our extremal perturbation masks are relatively more informative: by visualizing a series of area constrained masks, we show a ranking of regions and quantify their importance to the output prediction. We hope that further work on attribution heatmaps considers these criteria when designing future methods.

In addition to attribution method design, *metrics* for evaluating attribution techniques should also be faithful and informative. The earliest metrics for evaluating attribution heatmaps were based on evaluating performance on other computer vision tasks such as *weak localization* — *e.g.* the pointing game (J. Zhang et al., 2018). These metrics evaluate the quality of heatmaps based on how useful they are at identifying where an object is located in an image. While this is a useful task, it is an insufficient metric for measuring attribution quality; in particular, it

[4]The gradient of the linear model $\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ with respect to input $\boldsymbol{x}$ is the weights matrix $\mathbf{W}$; this is independent to the specific input $\boldsymbol{x}$ being explained.

can inadvertently penalize faithful attribution methods. Consider a model that has learned an incorrect but highly predictive correlation from the training data — *e.g.* "dumbbells" typically have "arms" in them (Mordvintsev et al., 2015). A perfectly faithful attribution method, when applied to such a biased model, might highlight regions containing "arms" when explaining the presence of "dumbbells" in a given image. Although such a method provides a perfectly accurate explanation of model behavior, it would be penalized in a weak localization task for not correctly highlighting "dumbbells." This example demonstrates the limitation of weak localization metrics to properly account for the fidelity of explanations when evaluating attribution methods. A number of other works have also highlighted this issue (Mordvintsev et al., 2015; Ribeiro et al., 2016; Lapuschkin et al., 2016; Kindermans et al., 2019; Lapuschkin et al., 2019).

One metric that does properly account for fidelity of explanations is the **deletion game**. Variants of it have been introduced over the past few years (Bach et al., 2015; Petsiuk et al., 2018; Kapishnikov et al., 2019). The main idea behind this metric is that pixels or regions are ranked in order of importance by an attribution heatmap. Then, these regions are iteratively deleted and the impact of the deletion is quantified, typically as the impact on classification accuracy (Bach et al., 2015; Petsiuk et al., 2018) or on image entropy (Kapishnikov et al., 2019).[5] However, this metric can be biased towards certain kinds of methods, such as techniques that generate highly pixelated visualizations[6] as well as methods like XRAI that follow a similar paradigm in their method design (*i.e.* a greedy algorithm for deleting or preserving regions).

Recently, several "sanity check" metrics have been introduced to better quantify the faithfulness of an attribution method. The **constant shift** test (Kindermans et al., 2019) checks whether a constant shift of all pixels in the input image, which does not affect a model's predictive ability, similarly does not affect an attribution method's produced heatmap. This check ensures that an attribution method is not sensitive to changes that do not affect model behavior. Another set of three tests check if an attribution method is sensitive (*i.e.* faithful) to the model, data distribution, and output class respectively being explained. The **model parameter randomization** test (Adebayo et al., 2018) checks whether randomizing a model's parameters affects an attribution method's produced heatmap. The **data randomization** test (Adebayo et al., 2018) checks whether randomly

[5] Another variant iteratively adds regions back to an image.

[6] As noted in (Kapishnikov et al., 2019), this is because a highly pixelated heatmap can span a larger region using the same number of pixels. Furthermore, deleting disjointed patches as is typically done can yield unnaturalistic artifacts that can also confound classification accuracy.

permuting the training labels affects attribution. Lastly, our own **class sensitivity** test (Rebuffi et al., 2020) (see appendix C.2) checks whether randomly permuting the output class being explained affects an attribution heatmap. These metrics should be viewed as unit tests for attribution methods; they do not positively evaluate attribution quality but rather test against the presence of undesirable qualities.

7.1.4 TorchRay: a package for reproducible research

The above discussion shows that there is ample room to further develop desiderata, "unit test"-like sanity checks, and further metrics to evaluate attribution quality. One of the barriers to this research development was an inability to easily reproduce other attribution methods and metrics. To that end, in collaboration with Facebook Research, we have developed and open-sourced the TorchRay package, which, in its initial release, implements a number of attribution methods (as well as a growing number of metrics) in such a way as to support reproducible research. Although a few similar packages exist (*i.e.* Captum[7] for PyTorch and saliency[8] for TensorFlow), they tend to be developed without benchmarking in mind (*i.e.* to simply generate attribution heatmaps).

Hopefully, TorchRay encourages more rigorous and reproducible research in the future.

7.1.5 Future work

In addition to further developing "unit test"-like sanity checks and benchmarks for assessing the quality of attribution heatmaps, there are a number of other promising future directions that follow naturally from our work (Fong et al., 2017; Fong et al., 2019a) as well as more broadly from the current state of attribution work.

The first is to *extend our perturbations framework* and combine it with other techniques to *explain every component of a model*. Most attribution methods for CNNs focus on the problem of *spatial attribution*, which is concerned with explaining what spatial regions are responsible for a model's prediction. In our works, we are primarily concerned with tackling the spatial attribution problem by perturbing the input image. That said, more work could be done with respect to spatial attribution at intermediate layers; this would allow us to study how information is filtered through the network.[9]

[7] `https://github.com/pytorch/captum`

[8] `https://github.com/PAIR-code/saliency`

[9] We explore the problem of spatial attribution at intermediate layers in Rebuffi et al., 2020, where we primarily focus on propagation-based attribution techniques.

In Fong et al., 2019a, we extend our perturbations framework to the problem of *channel attribution*, which is concerned with explaining which channels are responsible for a model's prediction. There, we attribute the channels after one layer in a network. However, our work could be extended to learn a series of channel attributions at various points throughout the network and thus combine *channel attribution* with *pathway attribution*, which is concerned with explaining which locations in a network (*i.e.* layers) are responsible for a model's prediction. Furthermore, we could also extend our perturbations framework to the under-explored problem of *weights attribution*, in which we highlight which network parameters are responsible for a model's decision. Such work could then be paired with recent work on visualizing the internal components of a model (Olah et al., 2020).

One of the challenges we faced in Fong et al., 2019a was making channel attributions interpretable. Unlike spatial attribution methods, which produce a heatmap that can be overlaid the input image, attribution of internal network components does not yield an intrinsically interpretable artifact. We visualize our channel attribution mask by generating a corresponding CAM/Grad-CAM-like (Zhou et al., 2016a; Selvaraju et al., 2017) spatial heatmap as well as a feature visualization (Olah et al., 2017) that highlighted the preferred stimuli of the channels in the attribution mask. That said, more work can and should be done to develop *visualization tools for non-spatial attribution*. Olah et al., 2018 make a relevant suggestion when they highlight the need for interpretability tools to be easily combined together like "building blocks." In the case on non-spatial attribution, more visualization "building blocks" are needed.

The second is to expand the *type of perturbations* we explore. In our work, we primarily explore the effects of spatial occlusions in images by replacing pixels with a reference signal (*i.e.* a blurred, randomly generated, or constant value pixel). However, there are many other image perturbations (*e.g.* scaling, cropping, warping) that can be used to tackle the attribution problem; many of these have been used for other computer vision applications such as self-supervised learning (Thewlis et al., 2017; Gidaris et al., 2018). Similar to the problem we faced when tackling channel attribution, one of the challenges when considering using these kinds of image transformations for attribution is how to render from them an interpretable and faithful explanation.[10] Thus, there is also ample opportunity to either explore novel interfaces or adapt existing ones (*e.g.* heatmaps) to explain effects of other perturbations. In our work on attributing channels (Fong et al., 2019a), we use a

[10]Spatial occlusions naturally translate well into a heatmap visualization.

constant value (*i.e.* 0) as a reference signal to perturb channels. Other reference signals should also be considered (or even generated, *i.e.* in a fashion similar to Chang et al., 2017; Chang et al., 2019). Often, the exploration of the reference signals (as done in Sturmfels et al., 2020) is itself quite fruitful for understanding CNNs.

The third is to leverage perturbations to *novel problems*. For instance, our perturbations framework could be extended to the problem of *uncertainty estimation*, which is concerned with quantifying the amount of uncertainty in a given measurement.[11] This can be done by applying perturbations to an input to a CNN and learning a distribution that best explains the observed output.[12,13] Another application of perturbations is to learn about global properties of a network. Morcos et al., 2018 and Zhou et al., 2018c can both be seen as using perturbations to ablate individual CNN hidden units in order to quantify (*i.e.* attribute) their importance for generalization performance (Morcos et al., 2018) and concept selectivity (Zhou et al., 2018c) respectively. Our Net2Vec work (Fong et al., 2018b) can also be viewed as learning a weighted perturbation over combinations of CNN hidden units to explain how a CNN layer is globally selective for a specific concept. Other directions in this vein include leveraging perturbations to attribute channels, pathways, and/or network weights for an entire output class or concept (as opposed to a single prediction). Yet another example is using perturbations to improve model robustness. We explore this in Fong et al., 2019c; there, we apply spatial occlusions during training in order to improve classification performance.

In summary, we have outlined three ways to build on our existing attribution work: 1. to explain other components of a model; 2. to leverage other kinds of perturbations; and 3. to apply perturbations to other problems.

7.2 Concept vectors

7.2.1 Related independent work to Net2Vec (Fong et al., 2018b)

Zhou et al., 2018b and Kim et al., 2018 independently introduced **IBD** (interpretable basis decomposition) and **TCAV** (testing with concept activation vectors)

[11] One way to view a typical CNN is that it predicts the *most likely* solution (*i.e.* the mean). If they were to also predict a range of solutions (*i.e.* a distribution over solutions) instead of a single solution, it would characterize the uncertainty (*i.e.* range or variance) of that prediction.

[12] This can be seen as an application of the explanation as meta-predictors framework we outlined in Fong et al., 2017.

[13] This can also be extended to do uncertainty estimation on other components in the same way as outlined above.

respectively; both of these techniques are methodologically similar to Fong et al., 2018b's Net2Vec concept vectors. However, they both differ in their application of concept vectors.

Zhou et al., 2018b primarily learn concept vectors in order to produce an instance explanation that attribute the semantic concepts that contribute to a CNN's prediction. For each attributed concept, a CAM-like (Zhou et al., 2016a) visualization is shown as well as a percentage of how much it contributed to the prediction. Zhou et al., 2018b also qualitatively show how concepts contribute to different output labels in a *Sankey diagram*.[14]

Similar to the latter application, Kim et al., 2018 primarily learn concept vectors to globally attribute concepts and their relevance to an output class. In particular, they introduce a relative concept vector, which allows them to compare the relative impact of two different concepts (*e.g.* are "stripes" or "dotted" patterns more relevant for the "zebra" class?). Kim et al., 2018 also report the accuracies of a few of their concept classifiers at different layers in order to support their argument that earlier layers tend to code for simpler concepts (*e.g.* colors) while later layers code for more complex ones (*e.g.* objects).

In contrast to the applications of IBD and TCAV to generate instance and global attribution explanations respectively, Net2Vec (Fong et al., 2018b) primarily focuses on comparing the representations of single filters and those of combinations of filters and argues that the single filter perspective is insufficient to fully describe semantic concepts. Fong et al., 2018b also highlight how viewing concept vectors as inhabiting an embedding space allows one to understand how a CNN understands concepts relative to one other as well as how different representations compare in their concept embeddings (*e.g.* what a CNN considers most similar to a "dog" concept is influenced by its training dataset and task, what a CNN consider supervised vs. unsupervised networks).

7.2.2 Future work

There are a few natural extensions to our Net2Vec work.

The first is to extend our Net2Vec to *learn distributions of concept vectors* (*i.e.* probabilistic probes). Currently, we learn a single concept vector; however, we can imagine a scenario where a variety of activation tensors code for the same concept. A simple and interpretable probabilistic model of the distribution of

[14] A *Sankey diagram* is a flow chart in which arrows are typically color coded and proportionally sized to reflect different magnitudes of contribution. See the Wikipedia article on "Sankey diagram."

activations associated to a single concept would be useful for better understanding how concepts are encoded.

The second is to *invert our Net2Vec setup*. Currently, we learn how to combine filters to be highly correlated to a single concept. We could also consider learning the inverse mapping: how to combine segmentation maps of concepts to be highly correlated to a single filter. This would allow us to better answer the question, "how many concepts can be 'packed' into a single filter?"

The third is to expand our analysis and leverage Net2Vec to *compare the differences* between model architectures (*e.g.* AlexNet vs. ResNet-50), supervision paradigms (*e.g.* supervised vs. self-supervised learning), and kinds of models (*e.g.* artificial CNN vs. human or other mammalian brain). While our work briefly outlined a few ways to do this using vector arithmetic (Bolukbasi et al., 2016) and representational similarity analysis (Kriegeskorte et al., 2008), much richer analysis can be done to better understand model differences using concept vectors. In particular, the development of powerful interfaces that would enable us to zoom in and out easily on analysis using concept vectors (*e.g.* from understanding differences in encoding a single concept vs. differences between whole concept embeddings) as well as to leverage other interpretability techniques would be particularly useful. Such tools and analysis would allow us to understand which concepts are more well defined.

7.3 Interactive visualizations

In comparison with other interactive works (fig. 2.18), there are few that are similar to our interactive similarity overlays in being a light-weight, "drop-in" exploratory tool for ML researchers and practitioners to understand the internal representation of CNNs (*i.e.* re-usable, white-box, interactive visualization). Many visual interfaces take varying amounts of effort to set up and are typically restricted to a specific framework, model implementation, and/or dataset format. In contrast, our method works for both PyTorch and TensorFlow and can easily be extended to work with any Python framework. We hope more work follows that provides a similar light-weight, "drop-in" experience to lower the barrier of usage. One example would be to develop an interactive, perturbation-based visualization that allows a human to visually perturb an input image and observe the effects.[15]

[15]This is somewhat similar to interactive visualizations for GANs (Zhu et al., 2016; Bau et al., 2019a).

There are a few natural extensions to our similarity overlays. The first is to use them to explore other kinds of tasks, models, and datasets — *e.g.* self-supervised tasks, recurrent models, medical imaging data, etc. The second is to use them to explore more hypotheses, such as to investigate whether CNNs exhibit the *gestalt phenomenon*[16] (Kim et al., 2019). The third is to further build the visualization with support for more features, such as color-coded positive and negative values[17] and support for interactive charts.

7.4 Interpretability research

Thus far, we have discussed potential research directions that are directly related to the work highlighted in this thesis. In this section, we take a step back and look more broadly at the interpretability research space and discuss other promising research directions that are currently under-explored.

We believe pursuing the following research avenues in the interpretability space would be highly useful to the development of AI:

1. explain other kinds of deep learning models beyond image classifiers;
2. develop deep learning models that are "interpretable-by-design;"
3. build techniques to "debug" and "fix" deep learning models that have learned incorrect correlations;
4. develop more metrics for evaluating interpretability techniques; and
5. build more intepretability tools for AI practitioners; in particular, more interactive visual tools for human exploration.

In the rest of this section, we discuss these research directions, in particular the first three, as the latter two have been discussed earlier in this chapter.

[16]The *gestalt phenomenon* refers to the human perceptual ability to perceive a whole object from an incomplete presentation of its parts (*e.g.* visual "auto-complete"). Optical illusions frequently rely on this phenomenon.

[17]Currently, we compute similarities of non-negative vectors (*i.e.* activations after ReLU layers); thus, our similarity scores are always non-negative.

7.4.1 Moving beyond image classifiers

Much work in interpretability research, including our own, has focused on building knowledge about deep learning by studying a particular kind of model: a CNN image classifier.

This is in part for historical reasons: deep learning regained popularity in the AI research community in 2012 when Krizhevsky et al., 2012 demonstrated how a CNN model (*i.e.* AlexNet) could yield superior performance on the image classification task when trained on a large dataset, like ImageNet (Russakovsky et al., 2015). Since then, there has been a flurry of research activity improving upon Krizhevsky et al., 2012. This has resulted in the development of better image classification models, such as VGG16 (Simonyan et al., 2015), GoogleNet (Szegedy et al., 2015), ResNet (He et al., 2016). Because standard implementations of these architectures were made available and supported in most major deep learning frameworks (*e.g.* TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017)), these architectures quickly became highly popular and easy-to-use. Thus, there was a significant amount of support for studying image classification models, compared to other tasks and domains, where the ecosystem was not as well developed and in which support was solidified behind a few canonical models.

Another reason image classification is often studied in interpretability research is due to the nature of the task and the large difference between the input and output data. Unlike object segmentation models, which produce interpretable masks that highlight the foreground pixels of a particular object, image classification models predict a single label that indicates the dominant object in the image. This classification label is not as interpretable as a dense segmentation mask. Thus, there is arguably more need to explain models trained on tasks like image clasification — *i.e.* ones that make concise predictions from high-dimensional input data.

That said, there are a number of other tasks and domains in which deep learning is making gains, and it would be highly valuable to develop interpretability techniques that explain the behavior and inner workings of models trained in those settings.[18] Here are just a few settings for which interpretability would be highly useful.

First, it would be useful to better understand models trained for *regression*, which predict a continuous value (*i.e.* a real number) as opposed to a discrete category as in classification. Due to the wide range of possible predictions, understanding regression

[18]This is the topic of an upcoming research workshop titled "XXAI: Extending Explainable AI Beyond Deep Models and Classifiers," which this author is co-organizing.

models would be highly useful and would likely require different techniques than those used to understand classification models.

Similarity models would also be interesting to study. Similarity models take as input two entities (*i.e.* two images) and either output whether or not the two inputs contain the same identity (*i.e.* two images of the same person) or predict a value (*i.e.* similarity score) that quantifies how similar the two images are. Due to its relative nature (*i.e.* outputs are relative and only make sense with respect to the inputs), it would be interesting to develop interpetability techniques for this domain.

Lastly, it would be useful to develop more understanding for deep learning models beyond convolutional ones, such as recurrent neural networks, deep generative models, and deep reinforcement learning models.

7.4.2 "Interpretable-by-design"

Another under-explored research direction is the development of deep learning models that are inherently interpretable by design.[19]

Much interpretablility work, including our own, has focused on understanding CNNs *post-hoc*, that is, after they have been trained. This is popular because much of the AI research community focuses on developing high-performance models and typically turn to intepretability literature once such a model has already been trained. That said, an arguably easier route to interpretable models would be to design models to be explicitly interpretable.

This is subtly different than simply designing models to produce explanations, as done in Hendricks et al., 2016; Z. Zhang et al., 2017; Huk Park et al., 2018. Much work on models that are interpretable by design has focused on models outside of the deep learning sphere (Lakkaraju et al., 2016; Kim et al., 2016). However, these models tend to not match deep learning in their capabilities. Thus, it would be highly beneficial to develop models that both inherently interpretable and build off of the success of deep learning. A few works have already made progress in these directions (X. Chen et al., 2016; Higgins et al., 2017; Q. Zhang et al., 2018; Brendel et al., 2019; C. Chen et al., 2019).

7.4.3 Model debugging

One application of interpretability techniques (*e.g.* attribution heatmaps) has been to identify systematic mistakes that models make. Because most machine learning tasks can be viewed as a prediction task, most machine learning models aim to

[19] See Rudin, 2019 for discussion on why "interpretable-by-design" models should be developed.

predict well based on the training data. Because training data is imperfect and often reflects the biases of humans (Buolamwini et al., 2018), models can learn highly predictive correlations that do not accurately reflect the causal relationship between inputs and outputs — *e.g.* images of "dumbbells" should have "arms" in them (Mordvintsev et al., 2015), the presence of snow is predictive of a husky dog (Ribeiro et al., 2016). A few works have highlighted this problem (Lapuschkin et al., 2016; Kindermans et al., 2019; Lapuschkin et al., 2019).

However, we currently have little recourse besides collecting a new dataset that corrects for the undesirable correlation and retraining a model on it. Thus, it would be highly useful if we could easily "debug" and "fix" deep learning models, much like how bugs in software are debugged and patched (*i.e.* fixed).[20] A few works have made progress in this direction (Khanna et al., 2018; Fuchs et al., 2019; Schulam et al., 2019; Kang et al., 2020).

7.4.4 Intrepretability metrics

As discussed in section 7.1.3, there is ample room to improve the metrics used for evaluating interpretability research. Similar to the earlier discussion on metrics for evaluating attribution heatmaps, there is little consensus in other interpretability topics (*e.g.* concept vectors) on the best ways to evaluate techniques, with most works building their own evaluation metrics or using qualitative examples to demonstrate the superiority of their visualizations (*e.g.* feature visualizations). Thus, further work can and should be done to better develop the desiderata of interpretability techniques as well as to critically evaluate them.

One potential conflict of interest in developing such metrics is that of businesses building products that use AI. As a community with close ties to industry, which stands to benefit greatly from research developments in AI, we should expect that our research community may have some shortcomings when it comes to rigorously evaluating our models. To that end, we should encourage as many diverse actors as possible to participate in the discussion of developing metrics (and more broadly, explainable AI). Furthermore, as a community, we should also consider setting up mechanisms for audits and transparency when it comes to model failures.

[20] This was the topic of a research machine learning workshop titled "Debugging machine learning models."

7.4.5 Interpretability tools for practitioners

Even though the focus of this thesis has primarily been research, translating research to be more broadly used is highly valuable. Similar to how medical research often translates into the clinic, we should aim to translate intepretability tools that would be broadly useful to any machine learning practitioner.

As previously discussed in section 7.1.4, much interpretability work is research-oriented: the majority of interpretability software consists of open-source repositories containing research code; these typically are not easy for non-expert, machine learning practitioners to use. A few works are more developer-oriented (*Captum* 2019; Wexler et al., 2019), yet there is ample space to build interpretability software toolkits to empower the non-expert developer.

In particular, we believe more work should be done in the space of interactive visualizations, in order to empower machine learning practitioners to easily explore trained models. As mentioned in section 2.4 and section 7.3, there are relatively few works that are similar to our interactive similarity overlays: Most work on interactive visualizations for deep learning is restricted to only explain a specific model (*i.e.* non-reusable), fail to visualize the internal dynamics of a model (*i.e.* are black-box methods), and/or use interactivity to navigate an interface (*i.e.* visual interfaces, as opposed to interactive visualizations). Thus, there is ample room for more work to be done in developing light-weight, easy-to-use tools that empower the machine learning practitioner (and researcher) to explore their model and gain insight about its behavior.

7.5 Conclusion

In summary, as AI is increasingly applied to high-impact yet high-risk applications, there is an increased need and desire for interpretability tools that aid humans in understanding such systems. In this dissertation, we present a number of novel methods for understanding CNNs. First, we present two principled methods for understanding what parts of an input image are responsible for a model's output decision (*i.e.* the *attribution* problem): **meaningful perturbations** and **extremal perturbations**. Second, we introduce **Net2Vec**, a novel paradigm for understanding how semantic concepts are encoded inside a CNN. Third, we present **interactive similarity overlays**, a new interactive visualization that enables AI researchers and developers to explore the internal representation of CNNs. Lastly, we outline some promising future directions in chapter 7. Just as medical diagnosis

is useless without treatment, we hope further work not only improves our ability to diagnose problems in CNNs but also empower us to repair them.

Appendices

A

Primer on relevant mathematical notation and concepts

Contents

In this primer, we describe essential mathematical notation and concepts used in this thesis. Note that notation might be slightly different in the chapters presented as papers (*i.e.* chapter 2 uses the notation presented here). Nevertheless, the concepts described here are highly relevant to understanding the material in this thesis and any other material that discusses CNNs.

This primer only assumes basic algebra (*e.g.* comfort with using variables) and is heavily based on Quantstart, 2017 and Brownlee, 2018.

A.1 Sets and their basic notation

A **set** is an *unordered*[1] collection of objects (*i.e.* order does not matter) with no duplicates. It can be either *finite* or *infinite*[2] in size. For example, the following are sets of the primary colors, a few points, and the natural numbers (*i.e.* set of positive integers) respectively:

$$\begin{aligned} A =& \{\text{red}, \text{blue}, \text{yellow}\}, \\ B =& \{(0,0), (0,1), (1,0), (1,1)\}, \\ C =& \{1, 2, 3, 4, \ldots\}. \end{aligned} \tag{A.1}$$

Sets are most frequently described using curly braces (*i.e.* $\{\ldots\}$) with elements delineated by commas and are often represented as uppercase, italicized variables (*i.e.* A, B, C), though they can be represented by other styles of uppercase variables.

A.1.1 Important sets

There are a few important sets that are represented by specific symbols. We have already described the set of natural numbers, which is denoted by $\mathbb{N} = \{1, 2, 3, \ldots\}$. Then there is the set of all integers (*i.e.* positive, negative, and zero), which is denoted by $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. Finally, there is the set of real numbers (*i.e.* all numbers that lie on a number line), which is given by $\mathbb{R}$; this includes

[1] We introduce ordered sets later in this section; assume sets are unordered by default.
[2] *i.e.* there can be an infinite number of objects in a set.

numbers like -1 (*i.e.* integers), $\frac{1}{3}$ (*i.e.* rational numbers that can be represented as a fraction), and π (*i.e.* irrational numbers).

A.1.2 Subsets

A **subset**[3] is a set which is entirely contained in another set and is denoted as follows:

$$A \subseteq B, \tag{A.2}$$

where $\subseteq$ can be read as "is a subset of". For instance, $\mathbb{N} \subseteq \mathbb{Z}$ but $\mathbb{N} \not\subseteq \mathbb{Z}$ (where $\not\subseteq$ reads as "is not a subset of"), because the natural numbers (*i.e.* positive integers) is a subset of all integers but the reverse relationship is not true.

There are several short-hand ways to describe subsets of real numbers (*i.e.* $\mathbb{R}$) that start and end at specific numbers. Consider all numbers between 0 and 1. If we wanted to describe this set of numbers and include 0 and 1, we would describe it using the following notation: $[0, 1]$, where square brackets denotes inclusion. If we wanted to exclude 0 and 1 from the set, we would describe it as follows: $(0, 1)$, where parentheses denotes exclusion. If we wanted to include 0 but exclude 1 from the set, we would describe it as follows: $[0, 1)$. Thus, the set of real numbers between a and b can be described using square brackets and/or parentheses, depending on whether each endpoint should be included or excluded from the set.

A.1.3 Set membership

To denote that an object can be found in a set, we use the inclusion symbol: $\in$. For example, $x \in \mathbb{R}$ states that x is in the set of real numbers (*i.e.* it is a real number).

A.1.4 Miscellaneous notation

To describe a statement that holds **for all** elements in a set, we use the following notation: $\forall x \in S \ldots$, which reads as "for all elements x in set S" as $\forall$ denotes "for all."

A.2 Scalars, vectors, matrices, and tensors

A.2.1 Scalars

A **scalar** is a single number and is represented by lowercase, italicized variables, such as $x = 1.2$ and $y = -2$. The important sets mentioned earlier (*i.e.* $\mathbb{R}$) are all sets of scalars.

[3] More precisely, A is a subset of B if and only if all elements in A are also in B.

A.2.2 Vectors

A **vector** is an *ordered* list of scalars (*a.k.a.* an *array*[4] of scalars). The most common vectors are points on an xy coordinate graph, such as $(1, 2)$; here, it's clear that position matters, as $(1, 2) \neq (2, 1)$. Vectors are represented by lowercase, boldface and italicized variables, such as $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{x} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \tag{A.3}$$

They can be represented as row vectors (*i.e.* a single row, like $\boldsymbol{x}$) or as column vectors (*i.e.* a single column, like $\boldsymbol{y}$) and with square brackets (*i.e.* $[\cdots]$) or parentheses (*i.e.* $(\cdots)$). In this thesis, we will use row and column vectors and both kinds of brackets interchangeably.

We can also describe them using set notation as follows: $\boldsymbol{x} \in \mathbb{N}^4, \boldsymbol{y} \in \mathbb{N}^3$. This denotes that $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors with 4 and 3 elements respectively and are comprised of natural numbers. We note the size of a vector by saying it is "n-dimensional" or of length n, where n denotes the number of elements in a vector — *i.e.* $\boldsymbol{x}$ is a 4-dimensional vector.

To reference an element at a specific position within a vector, we use scalar notation (*i.e.* lowercase, non-boldface) with a subscript to indicate the index of the position:

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \cdots \\ z_n \end{bmatrix}. \tag{A.4}$$

In the above example, z_i refers to the element in the i-th position. In the earlier examples, $x_3 = 3$ and $y_2 = 2$.

A.2.2.1 Vector arithmetic

To add two vectors, they must be the same size. Then, for every position in the vector, the two corresponding elements are added together. Given two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^N$, adding the two vectors would result in the following vector $\boldsymbol{c}$:

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \cdots \\ a_n + b_n \end{bmatrix}. \tag{A.5}$$

[4] In computer science, an *array* is a fixed-length, ordered list.

Similarly, *element-wise multiplication* takes two vectors of the same size and multiplies together each of their corresponding elements to produce another vector of the same size. Given two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^N$, we denote element-wise multiplication using the Hadamard operator $\odot$ as follows:

$$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b} = \begin{bmatrix} a_1 \times b_1 \\ a_2 \times b_2 \\ \cdots \\ a_n \times b_n \end{bmatrix} \tag{A.6}$$

The resulting vector is also known as the *Hadamard product*.

To multiply a vector with a scalar every element in the vector is multiplied by the scalar. Given a vector $\boldsymbol{c} \in \mathbb{R}^N$ and a scalar $s \in \mathbb{R}$, multiplying (*i.e.* scaling) $\boldsymbol{c}$ by s sould result in the following vector $\boldsymbol{d}$:

$$\boldsymbol{d} = s\boldsymbol{c} = \begin{bmatrix} s \times c_1 \\ s \times c_2 \\ \cdots \\ s \times c_n \end{bmatrix}. \tag{A.7}$$

A.2.3 Matrices

A matrix is a 2-dimensional[5] rectangular array of scalars and represented by uppercase and boldface variables, such as $\mathbf{X}$ and $\mathbf{Y}$:

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}. \tag{A.8}$$

Using set notation, we would say $\mathbf{X} \in \mathbb{N}^{2\times3}$ and $\mathbf{Y} \in \mathbb{N}^{3\times2}$, where the superscript $M \times N$ denotes a matrix with M rows and N columns.

To reference elements within a matrix, we use scalar notation (*i.e.* lowercase, non-boldface) with subscripts denoting the row and column of the element:

$$\mathbf{Z} = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m,1} & z_{m,2} & \cdots & z_{m,n} \end{bmatrix}. \tag{A.9}$$

In the above example, $z_{i,j}$ refers to the element at in the i-th row and j-th column of the matrix $\mathbf{Z} \in \mathbb{R}^{M\times N}$. In the earlier examples, $x_{2,1} = 4$ and $y_{2,1} = 3$.

[5]The *dimensionality* of an array denotes its geometric shape. A vector is a 1-dimensional array because it expands *along* one dimension (*i.e.* either down a column vector or across a row vector). This is in contrast to the dimensionality of a vector, which denotes the number of elements in it.

A.2.4 Tensors

A **tensor** is a generalization of scalars, vectors, and matrices and can be an array of any *dimensionality*.[6] In the context of CNNs, 3D and 4D tensors are often used. An image can be viewed as a 3D tensor, where the 3rd dimension corresponds to the color channel. A batch of images (*i.e.* ordered set of images) can be viewed as a 4D tensor, where the 4th dimension corresponds to the index of an image.

In this thesis, we represent tensors such as an image tensor as follows: $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$, where the tensor variable $\boldsymbol{x}$ is represented in the same way as a vector (*i.e.* lowercase, boldface, italicized) and the number of elements being multiplied in the superscript of a set (*i.e.* 3 elements: 3, H, and W) denotes the order of the tensor. Conceptually, a 3D tensor can be imagined as a cube[7] with a specific depth, height, and width. An element in a 3D tensor is given by $x_{i,j,k}$, where i, j, k denotes the index of the channel, row, and column respectively.

We represent a 4D tensor (*e.g.* batch of images) as follows: $\boldsymbol{x} \in \mathbb{R}^{B\times C\times H\times W}$, where B denotes the batch size (*i.e.* number of images), C denotes the number of channels ($C = 3$ for RGB images), H denotes the height (*i.e.* number of rows), and W denotes the width (*i.e.* number of columns). An element in a 4th-order tensor is given by $x_{i,j,k,l}$, where i, j, k, l denotes the index of the batch element, channel, row, and column respectively.

In this thesis, we will always order the dimensions of tensors with order greater than 1 as follows: 1. batch (if given); 2. channel (if given); 3. row; 4. column.

Arithmetic operations on tensors (*e.g.* addition, element-wise multiplication, scalar multiplication) work in the same way as those on vectors; matrix multiplication can also be generalized to larger-order tensors.

A.3 Sequences

Mathematical operators are often applied to sequences (*i.e.* sum up all the elements in a vector). In this section, we describe common notation used to succintly describe sequences.

[6] The *dimensionality* of an array is also known as the *order* of an array — *i.e.* a 3D tensor can also be described as a 3rd-order tensor.

[7] More precisely, as a *cuboid*.

A.3.1 Ordered sets

To describe an *ordered set*[8] with N elements, we use the following notation:

$$\{x_i\}_{i=1}^{N} = \{x_1, x_2, \ldots, x_{N-1}, x_N\}, \tag{A.10}$$

where x_i refers to the i-th element in the set, and the sub and super-script detail the variable representing the index (*i.e.* i), its starting value (*i.e.* 1) and its ending value (*i.e.* N). An ordered set can contain duplicates, that is, $x_i = x_j$ where $i \neq j$ is allowed. In this thesis, assume that sets are unordered by default, unless explicitly stated or denoted with this index notation.

A.3.2 Summing sequences

To describe summing along a sequence, we use the following notation:

$$y = \sum_{i=1}^{N} x_i = x_1 + x_2 + \ldots + x_{n-1} + x_n, \tag{A.11}$$

where x_i denotes the i-the element in the sequence and the uppercase Greek letter Sigma (*i.e.* Σ) represents the sum operation.

A.3.3 Multiplying sequences

To describe taking the product (*i.e.* multiplying together) of elements of a sequence, we use the following notation:

$$y = \prod_{i=1}^{N} x_i = x_1 \times x_2 \times \ldots x_{n-1} \times x_n, \tag{A.12}$$

where the uppercase Greek letter Pi (Π) represents the product operation.

A.4 Tensor operations

In the context of deep learning, we often describe custom operations to tensors (*i.e.* passing a tensor through a layer). We use function notation to describe operations: $f : D \to R$, where f denotes the function (*i.e.* operation), D denotes the *domain* (*i.e.* the set of possible input values), and R denotes the *range* (*i.e.* the set of possible output values).

[8] For the sake of simplicity, we abuse notation and describe what is typically known as a *indexed family* as an *ordered set* — *i.e.* a collection of objects in which each object has an index. See the Wikipedia article on "Indexed family."

The operations we consider typically take one input and produce one output (though some CNN operations can take multiple inputs and produce multiple outputs). We will most often describe this as follows:

$$\boldsymbol{y} = f(\boldsymbol{x}), \tag{A.13}$$

where $\boldsymbol{x}$ denotes the input tensor the operation f and $\boldsymbol{y}$ denotes the output of operation f when applied to $\boldsymbol{x}$.

A.4.1 Linear combinations

A common operation that is frequently used in neural networks is computing the **linear combination** of terms. A linear combination refers to the operation that, when given a set of tensors (*i.e.* $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$) and a set of scalars (*i.e.* $\{a, b, c\}$), multiplies each term with a constant and sums the result:

$$a\boldsymbol{x} + b\boldsymbol{y} + c\boldsymbol{z}, \tag{A.14}$$

In the context of deep neural networks, the terms typically correspond to the values in an input tensor and the constants to the learned parameters (*i.e.* weights).

A.4.2 Dot product

Another common operation that is closely related to linear combinations is taking the **dot product** between two tensors (*i.e.* $\boldsymbol{x}$ and $\boldsymbol{y}$) of the same size. The dot product is the sum of the products of the corresponding entries in two sequences (*i.e.* the Hadamard product) and is denoted using the $\cdot$ operator. For example, the dot product of two 3D tensors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{C \times H \times W}$ is given by

$$z = \boldsymbol{x} \cdot \boldsymbol{y} = \sum \boldsymbol{x} \odot \boldsymbol{y} = \sum_{c=1}^{C} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{c,i,j} \cdot y_{c,i,j}, \tag{A.15}$$

In the context of convolutional weights, an output value is computed by taking the dot product of a set of weights and a subset of input values of the same size.

A.4.3 Matrix multiplication

Matrix multiplication is also frequently used in deep learning. In order to do multiply two matrices, $\mathbf{A} \in \mathbb{R}^{M \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times P}$, the number of columns in A needs to equal the number of rows in $\mathbf{B}$ (*i.e.* N, the inner dimensions need to match). The dimensions of resulting matrix $\mathbf{C} \in \mathbb{R}^{M \times P}$ is given by the outer

dimensions of the two matrices (*i.e.* number of rows in $\mathbf{A}$ and number of columns in $\mathbf{B}$). Then, an element in the resulting matrix $\mathbf{C} = \mathbf{AB}$ is computed as the dot product of a row in $\mathbf{A}$ and a column in $\mathbf{B}$:

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j} \tag{A.16}$$

Note that, by definition, the order of the matrices being multiplied matters, that is $\mathbf{AB} \neq \mathbf{BA}$. Here are the matrices with their element notation:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} \dots & a_{m,n} & \end{bmatrix} \begin{bmatrix} b_{1,1} & \dots & b_{1,p} \\ b_{2,1} & \dots & b_{2,p} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,p} \end{bmatrix} = \begin{bmatrix} c_{1,1} & \dots & c_{1,p} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \dots & c_{m,p} \end{bmatrix}. \tag{A.17}$$

Now, we can work out the following example. Given the following matrices,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix} \tag{A.18}$$

The result of multiplying the matrices $\mathbf{C} = \mathbf{AB}$ is given as

$$\mathbf{C} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 & 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 \\ 2 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 & 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 4 \end{bmatrix} = \begin{bmatrix} 14 & 20 \\ 20 & 29 \end{bmatrix} \tag{A.19}$$

A.4.4 Matrix tranpose

Often, the **transpose** of a matrix is taken before doing matrix multiplication in order to make the inner dimensions align. A matrix transpose, denoted as $\mathbf{M}^T \in \mathbb{R}^{N \times M}$ is a version of another matrix $\mathbf{M} \in \mathbb{R}^{M \times N}$ in which its rows and columns have been swapped.

Here is an example of a matrix and its transpose:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \mathbf{M}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}. \tag{A.20}$$

A.5 Random numbers and variables

For a more detailed treatment on basic probability, see Blitzstein et al., 2019.

Many times, we desire to generate a random number. There are two main *distributions*[9] from which we typically *sample* (*i.e.* draw or select) random numbers.

[9] A *distribution* is a succinct description of all possible elements and the likelihoods (*i.e.* probabilities) with which they can be sampled.

Basic notation. We represent a random number and the distribution it is drawn from as follows:

$$X \sim \mathcal{D}, \tag{A.21}$$

where $\sim$ represents drawing X (on its left-hand) from $\mathcal{D}$ (on its right-hand), X denotes a random variable that represents the random number drawn, and $\mathcal{D}$ represents the distribution being drawn from.

A.5.1 Uniform distribution

The **uniform distribution** is one of the most basic random distribution and describes drawing with equal likelihood (*i.e.* "drawing uniformly") any option from a set of possible options.

We most typically deal with a uniform distribution of a subset of the real numbers (*i.e.* $\mathbb{R}$) and can describe it in the same way we describe a range of real-numbers starting at the minimal point a and ending at the maximal point b, with the option of including or excluding the endpoints respectively: $[a, b]$ or (a, b).

To describe the Uniform distribution succinctly, we use the following notation:

$$\text{Uniform}(a, b) \text{ or } \mathcal{U}(0, 1), \tag{A.22}$$

where kind of bracket used denotes inclusion or exclusion of the endpoints.

Then, the following denotes a random variable X drawn from $[0, 1]$: $X \sim \mathcal{U}[0, 1]$.

One way to describe the a distribution is via a special function, known as a **probability density function (PDF)**, whose output describes the relative likelihood that the value of a random variable drawn from said distribution would be equal to the input value to the function.

For the uniform distribution, its PDF is given as follows:

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{for } a \leq x \leq b, and \\ 0, & \text{for } x < a \text{ or } x > b. \end{cases} \tag{A.23}$$

This PDF function shows that all possibilities between a and b are equally likely.

A.5.2 Normal distribution

The **normal distribution** (*a.k.a.* **gaussian** distribution) is another basic random distribution and is most succinctly preciesly described by its PDF function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, \tag{A.24}$$

where $\mu \in \mathbb{R}$ (lowercase Greek letter mu) is a real number that describes the *mean* (*i.e.* average) of the distribution and $\sigma \in \mathbb{R}^+$ (lowercase Greek letter sigma) is a positive, real number that describes the *standard deviation* of the distribution.

A plot of the PDF function $f(x)$ looks like a bell-shaped curve, where the middle of the bell is at mean μ and the standard deviation σ controls how spread out the bell-shape curve is.

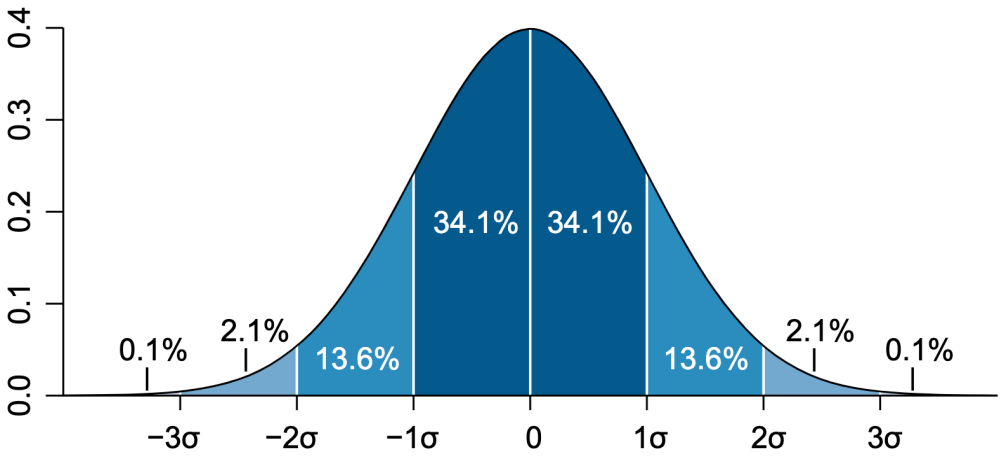


Figure A.1: PDF of normal distribution with $\mu = 0$. The percentages shown highlight the percentage of the total area under the curve that is contained in a given portion of the curve. Notice how these percentages demonstrate the 68-95-99.7 rule. Reproduced image by M. W. Toews under CC BY 2.5.

More precisely, the standard deviation σ is the width on both sides of the mean that defines an input range of $(\mu - \sigma, \mu + \sigma)$. The portion of the area under the curve that falls within that input range (*i.e.* within one standard deviation, $\pm\sigma$) is 68% of the total *area under the curve* (AUC). The corresponding portion that falls within two standard deviations (*i.e.* input range of $(\mu - 2\sigma, \mu + 2\sigma)$) is 95% of the total AUC, and the corresponding portion for 3 standard deviations (*i.e.* input range of $(\mu - 3\sigma, \mu + 3\sigma)$) is 99.7%. This property of normal distributions holds regardless of specific μ and σ values and is known as the *68-95-99.7 rule*.

Intuitively, samples drawn from a normal distribution are more likely to be close to its mean μ than far away from it, with values greater than 3 standard deviations (*i.e.* σ) away less than 0.3% likely to be drawn.

Notation describing the normal distribution is as follows

$$\text{Normal}(\mu, \sigma) \text{ or } \mathcal{N}(\mu, \sigma), \tag{A.25}$$

where μ and σ denote its mean and standard deviation respectively.

Standard normal distribution. Also known as the **unit normal distribution**, the **standard normal distribution** is the Normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$ — *i.e.* $\mathcal{N}(0, 1)$) — and is frequently used, hence its special designation.

A.5.3 Bernoulli distribution

So far, we have only discussed *continuous* distributions that span a range of real numbers. However, we can also have *discrete distributions* which present a set of categories each with a different likelihood of getting drawn. Let us consider the simplest discrete distribution: the **Bernoulli distribution**. Intuitively, the Bernoulli distribution can be likened to a heads or tails coin flip of a weighted coin. Let p be the probability of seeing heads; then, the probability of seeing tails is $1 - p$. Formally, if X is a Bernoulli random variable with parameter p, then

$$P(X = 1) = p; P(X = 0) = 1 - p, \tag{A.26}$$

where $X = 1$ corresponds to the "heads" outcome in the coin flip analogy. Then, a fair coin could be represented as a Bernoulli with $p = 0.5$.

A.5.4 Expectation

An high-level understanding how to compute an expectation is helpful for understanding the expected gradients method (Sturmfels et al., 2020), which we discuss in section 2.2.1.2 as part of our literature review. In this section, we provide such a high-level explanation.[10]

The **expectation** of a random variable is the weighted average of all possible options. Intuitively, it is the arithmetic mean of a distribution.

For a finite, discrete random variable X with a finite set of possibilities: $x_1, x_2, \ldots, x_n$, its expectation is as follows:

$$\mathbb{E}[X] = \sum_{i=1}^{n} x_i P(X = x_i) = x_1 P(X = x_1) + \ldots + x_n P(X = x_n) \tag{A.27}$$

Consider a random variable X that is Bernoulli distribution with parameter p. Using eq. (A.27), we can compute the expectation of a Bernoulli as follows:

$$\mathbb{E}[X] = 0P(X = 0) + 1P(X = 1) = p \tag{A.28}$$

[10] See the Wikipedia article on "Expected value" for more information.

For the other two continuous distributions we covered, we will use the intuitive definition of the arithmetic mean to introduce their expectations. The expectation for a normally distributed random variable is given by μ:

$$\mathbb{E}[X] = \mu. \tag{A.29}$$

For a uniformally distributed random variable $X \sim \text{Uniform}(a, b)$, its expectation is given by

$$\mathbb{E}[X] = \frac{b-a}{2}. \tag{A.30}$$

Intuitively, this makes sense: the mean of a uniformally distributed random variable is the midpoint between the endpoints of the distribution.

A.5.5 Independent and identically distributed random variables

So far, we described drawing real-valued scalars from the uniform and normal distribution. However, in the context of deep learning, we often deal with whole tensors that are randomly generated. The elements of such tensors can be described most commonly as being **independent and identically distributed (i.i.d.)** random variables.

Two random variables X and Y are *identically distributed* if they are drawn from the same distribution (*i.e.* they have the same PDF function). Given the following random variables:

$$X \sim \mathcal{N}(0,1), Y \sim \mathcal{N}(0,1), Z \sim \mathcal{N}(0,2), \tag{A.31}$$

X and Y are identically distributed but X and Z are not (because they have different standard deviations σ, resulting in different PDF functions).

Two random variables X and Y are *independent* if the processes that generate them are indepedent (*e.g.* not conditioned) on each other. For example, snowy conditions and the temperatures are not independent events because snowy conditions necessitates a temperature below freezing point (*i.e.* snow is conditioned on temperature). One way to formally describe independent events is that the probability of both events happening is equal the product of the probability of both events happening on its own. This can be described mathematically as follows:

$$f_{X,Y}(x, y) = f_X(x) f_Y(y), \text{ for all } x, y, \tag{A.32}$$

where f_X and f_Y are the PDFs for random variables X and Y respectively and $f_{X,Y}$ is the joint PDF that describes the likelihoods of how pairs of possibilities (x, y) occur.

In our context, we often sample a tensor whose elements are i.i.d. from the same distribution. We use the following notation to describe this:

$$\boldsymbol{x} \overset{\text{i.i.d.}}{\sim} D, \tag{A.33}$$

where $\boldsymbol{x}$ is a tensor whose elements are i.i.d. sampled from distribution D. Thus, $\boldsymbol{x} \overset{\text{i.i.d.}}{\sim} \mathcal{U}[0, 1]$ denotes a tensor whose elements are all uniformly drawn from the range $[0, 1]$.

A.6 Basic calculus

In this section, we cover some basic concepts from calculus[11] that are helpful for understanding how CNNs work. To read this thesis, one does not need to fully understand these concepts, but simply need to understand at a high-lvel what computing the "gradient" or "integral" means. Thus, we focus on providing an intuititive explanation of these concepts and refer the reader to other sources to fully understand them.

A.6.1 Differentiation: computing a gradient

A **derivative** or **gradient** of a function f at a point x refers to the *slope* (*a.k.a.* the "rate of change") of the function at that point (see fig. A.2).[12]

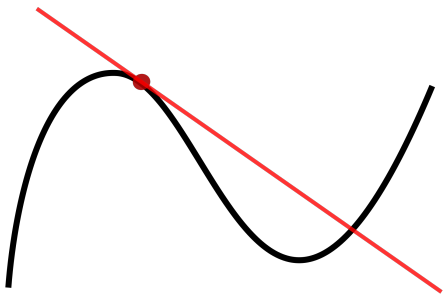


Figure A.2: Gradient of a function. This diagram visualizes a function (black curve) at the specific point (red dot). The gradient of the function at the point is given by the slope of the red line. This image is by Jacj at English Wikipedia and is available in the public domain.

[11] Calculus comes from the Latin word meaning "small stone" and primarily concerns studying how small changes affect a function. See this article on "Introduction to Calculus" for an accessible introduction.

[12] This section is based on this article titled "Introduction to Derivatives."

Slope between two points. On an xy-plot (*i.e.* a Cartesian plot), the slope of a function is expressed as follows (fig. A.3):

$$\text{slope} = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x}, \tag{A.34}$$

where the upper-case Greek symbol delta (Δ) is read as "change in" and represents the difference between two quantities. Formally, the slope m between two points (x_1, y_1) and (x_2, y_2) can be calculated as follows:

$$m = \frac{y_2 - y_1}{x_2 - x_1}. \tag{A.35}$$

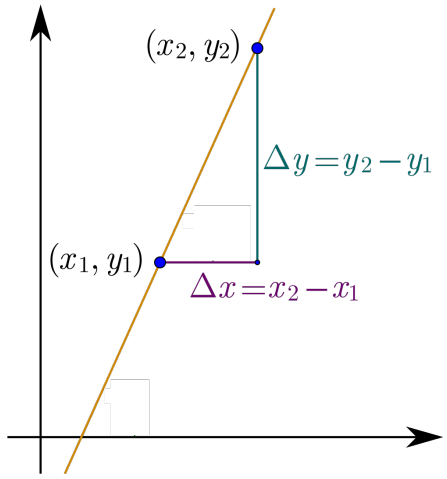


Figure A.3: Slope between two points. The slope between two points can be calculated as the change in y over the change of x — *i.e.* $\Delta y/\Delta x$, see eq. (A.35). This image was originally created by Maschen; we modified and reproduced it via CC BY-SA 3.0.

For a motivational example, consider how we describe the speed of cars. Suppose you drive 20 miles in 20 minutes. We would describe that speed as 60 miles per hour. This is actually a slope and describes the average rate of change (*i.e.* change of miles per hour) of a vehicle and is computed as follows:

$$\frac{20 \text{ miles}}{\frac{1}{3} \text{ hour}} = 60 \text{ mph}. \tag{A.36}$$

Gradient at a single point. To find the slope at a single point, we consider a very small change in x $x + \Delta x$ — affects the output of a scalar function f. We can express the slope between x and $x + \Delta x$ as follows:

$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{A.37}$$

Then, to find a slope at x, we consider what happens when Δx shrinks to become infinitesimally small, that is, when Δx goes to 0 (*i.e.* $\Delta x \to 0$). This is the gradient or derivative of a function f at point x (see fig. A.4).

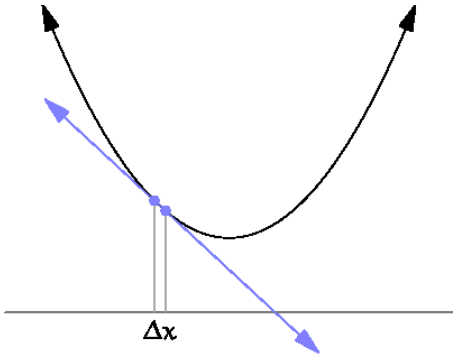


Figure A.4: Δx. Intuitively, the gradient of a function at a point x is the slope of the function between x and $x + \Delta x$ when Δx is infinitesimally small (*i.e.* when Δx goes to 0). This image is part of an animated image originally created by Wikipedia user Brnbrnz and is reproduced it via CC BY-SA 4.0.

Here, we have shown the gradient of a scalar y with respect to another scalar x. In the context of CNNs, a gradient of a scalar is typically computed with respect to a multi-dimensional tensor — *i.e.* the gradient of a loss function with respect to a network's weight parameters, which are typically stored as tensors.

In this thesis, we represent a gradient of a scalar y with respect to a tensor $\boldsymbol{x}$ as follows:

$$\frac{\partial y}{\partial \boldsymbol{x}}, \tag{A.38}$$

where the gradient would have the same shape as $\boldsymbol{x}$.

A.6.2 Integration: computing an integral

An high-level understanding of integrals is helpful for understanding the integrated gradients method (Sundararajan et al., 2017), which we discuss in section 2.2.1.2 as part of our literature review.[13]

Intuitively, an **integral** can be viewed as the "area under a curve," and a *definite integral* is the area under a curve in between points a and b. Figure A.5 visualizes this interpretation. Given a scalar function f (*i.e.* the "curve"), the definite integral of f between a and b is the sum of the colored areas between the function and the x-axis. Areas where $f(x) < 0$ contribute negatively to the integral, while areas where $f(x) > 0$ contribute positively to the integral.

[13]For an accessible introduction to integration, see this article on "Introduction to Integration."

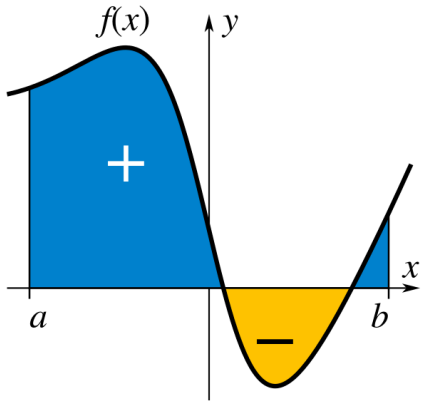


Figure A.5: Example of an integral. The definite integral of a function $f(x)$ between points a and b is the sum of the signed areas under the "curve" of the function. For the parts where $f(x) < 0$ (in yellow), those areas contribute negatively to the integral. This image is by Wikipedia user KSmrq and is reproduced via CC BY-SA 3.0.

One way to think about approximating an integral is to be summing up "slices" underneath a curve. Let us consider integrating the function $f(x) = \sqrt{x}$ between 0 and 1. We can do this by "slicing up" the function as done in fig. A.6. Then,

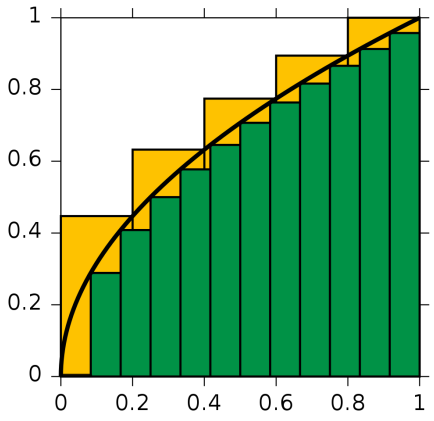


Figure A.6: Approximating an integral. Here is an example of how the integral of the function $f(x) = \sqrt{x}$ from $x = 0$ to $x = 1$ can be approximated. This image is by Wikipedia user KSmrq and is reproduced via CC BY-SA 3.0.

the integral is approximately equal to the sum of the yellow slices (or the sum of the green slices). Concretely, let us work out the example shown in yellow. Here, we evaluate $f(x)$ at the following points: $0.2, 0.4, 0.6, 0.8, 1$ (*i.e.* in increments of $\Delta x = 0.2$). To compute the area of one slice, we multiply the height of the slice (*i.e.* $f(x)$) with its width (*i.e.* Δx). Then, we can approximate the integral

by summing up the areas of all slices:

$$0.2(f(0.2) + f(0.4) + f(0.6) + f(0.8) + f(1.0)) = 0.7497. \tag{A.39}$$

To compute an integral exactly, we consider what would happen if the slices became infinitesimally small (*i.e.* $\Delta x \to 0$). Formally, we write a definite integral as follows:

$$c = \int_a^b f(x)dx, \tag{A.40}$$

where a and b denote where to start and end integrating, $f(x)$ denotes the function to integrate, and dx denotes we are considering slices along x (*i.e.* we start integrating at $x = a$ and stop at $x = b$).

An integral can be viewed as the inverse of the derivative. To illustrate the relationship between an integral and a derivative (*i.e.* a rate of change), let us consider the vehicle speed example again. Let $f(x)$ denote the speed of our vehicle at time x (in hours). Then, if we are interested in the distance we have travelled in the first 1 hour of driving, we can express it as follows: $\int_0^1 f(x)dx$. Suppose we maintained an expect speed of 20 m.p.h for the first 20 minutes, 40 m.p.h. for the second 20 minutes, and 60 m.p.h. for the third 20 minute-segment. Then, we can write $f(x)$ as follows:

$$f(x) = \begin{cases} 20 & \text{for } 0 \leq x \leq \frac{1}{3}, \\ 40 & \text{for } \frac{1}{3} \leq x \leq \frac{2}{3}, \\ 60 & \text{for } \frac{2}{3} \leq x \leq 1. \end{cases} \tag{A.41}$$

Now, we can compute our total distance in an hour as follows:

$$\int_0^1 f(x)dx = \frac{1}{3}(20 + 40 + 60) = 40 \text{ miles}. \tag{A.42}$$

A.7 Notation for Convolutional Neural Networks (CNNs)

In this thesis, we describe an object classification CNN as the following function:

$$\Phi : \mathbb{R}^{3\times H\times W} \to [0, 1]^C, \tag{A.43}$$

where Φ (*i.e.* uppercase Greek letter Phi) denotes the CNN, $\mathbb{R}^{3\times H\times W}$ specifies the input domain as the set of 3D tensors with 3 color channels, H rows, and W columns that contain real numbers (*i.e.* RGB images), and $[0, 1]^C$ denotes the

output range the set of vectors of length C with elements ranging from 0 to 1 inclusive. The output of the CNN can be viewed as vector containing probabilities, where each element represents the probability that a specific object category is the dominant object in the image.

To denote that the values of the network's output vector has not been normalized to range between 0 and 1 (*i.e.* does not represent probabilities), we would describe it as follows:

$$\Phi : \mathbb{R}^{3\times H\times W} \to \mathbb{R}^{C}. \tag{A.44}$$

A.7.1 Partial networks

To describe the set of operations (*i.e.* layers) from one point in the network Φ to another (*i.e.* from layer l_1 exclusive to layer l_2 inclusive) we use the following notation:

$$\Phi_{l_1}^{l_2}, \tag{A.45}$$

where the subscript denotes that the output of layer l_1 is the input to our partial network $\Phi_{l_1}^{l_2}$ and the superscript denotes that l_2 is the last layer applied to produce the output of the partial network.

If we want to describe the network up to a certain point (*i.e.* starting from the first layer to layer l), we use the following notation:

$$\Phi^{l}, \tag{A.46}$$

where the superscript denotes the last layer applied to produce this partial network's output.

Finally, to describe selecting an element from the output tensor produced by the network Φ, we use the following notation:

$$\Phi_c : \mathbb{R}^{3\times H\times W} \to \mathbb{R}, \tag{A.47}$$

where c denotes the index of a particular object category whose output we are interested in.

A.7.2 Inputs, outputs, and intermediate tensors

A CNN can often be described as applying a series of layers (*i.e.* operations) that each take as input a tensor and produce as output another tensor, that may or may not have a different shape (*i.e.* the domain and range of the operation may be different). In this thesis, we will usually describe the input tensor to a CNN Φ as

$\boldsymbol{x}$ and the output tensor produced by Φ as $\hat{\boldsymbol{y}}$, and an intermediate tensor (*i.e.* the output of a layer that is not the last layer in Φ) as $\boldsymbol{z}$. We use the hat symbol on top of $\boldsymbol{y}$ (*i.e.* $\hat{\boldsymbol{y}}$, read as "y-hat") to denote an estimate of $\boldsymbol{y}$ (*i.e.* the target or ground-truth label). Intermediate tensors are also known as **activations** or **activation tensors**.

B

Primer on convolutional neural networks

Contents

In this section, we provide a brief primer on deep learning, with a particular emphasis on explaining the basics for training an object classification CNN. This primer is primarily for readers with minimal background knowledge. For a primer on the mathematical notation and concepts used, see appendix A. In particular, see appendix A.7 for a detailed description of the notation we use to describe CNNs.

B.1 Machine learning set-up

A machine learning set-up is primarily defined by the following four ingredients: the *model* being trained as well as the *data*, *task*, and *training procedure* used.

B.1.1 Data

The *training data* is most often defined as pairs of inputs and outputs[1] (*i.e.* $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$), where a model's aim is to predict an output from its corresponding input. For object classification, the inputs are RGB images (*i.e.* $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$) and the outputs are labels of the most dominant object in the corresponding images (*i.e.* "sheepdog", "sea snake", "soup bowl", "alps", etc.). The output label is also known as the **ground truth** label. The output label for object classification is usually represented as a *one-hot vector*[2] of length C, where C denotes the number of object categories (*i.e.* $\boldsymbol{y} \in \mathbb{R}^{C}$) and the indices of the vector denote different object categories. For a given input-output pair (*i.e.* $(\boldsymbol{x}, \boldsymbol{y})$), the index of the output vector (*i.e.* $\boldsymbol{y}$) containing 1 denotes the most dominant object in the input image (*i.e.* $\boldsymbol{x}$).

B.1.2 Task

The format of the training data is deeply related to the training task, which is typically an informal description of the prediction task that the model is being trained for.

B.1.2.1 Object classification

For example, *object classification* refers to the task of predicting the dominant object category of an image, while *colorization* (R. Zhang et al., 2016) refers to predicting a color image from a black-and-white version of it. More formally, the task is typically defined by the training data and the *loss function* used to train the model. The majority of machine learning tasks aim to minimize a loss function, which captures a notion of error or incorrect behavior and can be *optimized*.[3]

B.1.2.2 Cross-entropy loss

For classification problems, the *cross-entropy loss* (*a.k.a. log loss*) function is typically used.[4][5] In its simplest form, it measures the performance of a classification model

[1] When the outputs require human annotation (*i.e.* labeling), this set-up is known as *supervised learning*. When the outputs do not require human annotation (*i.e.* they are free), this set-up is known as *self-supervised learning*. An example of a self-supervised set-up is predicting a color image from a black-and-white version of it (R. Zhang et al., 2016).

[2] A *one-hot vector* is a vector which is filled with zeros except at one position, where it is filled with a 1 (*e.g.* $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}$).

[3] *Optimization* refers to the selection of the best solution from a set of possible solutions. See the Wikipedia article on "Mathematical optimization".

[4] This paragraph is paraphrased from (*Loss Functions* 2017).

[5] See this explanation for a more thorough and accessible, visual treatment of the cross-entropy loss function.

whose output is a probability between 0 and 1 (*i.e.* a binary classifier, in which positive examples should be predicted as 1 and negative examples as 0). The cross-entropy loss is large when the predicted probability is relatively far away from the true (*a.k.a.* ground-truth) label and is small when the predicted probability is relatively close to the true label. Thus a perfect model would have a loss of 0 and the goal of the model is to minimize the loss so that its predictions are as close as possible to the true labels.

Mathematically, for a binary classifier for which the label is either 0 or 1 (*i.e.* $y \in \{0, 1\}$) and its prediction is between 0 and 1 (*i.e.* $\hat{y} \in [0, 1]$), the cross-entropy loss function is defined as follows:

$$\mathcal{L} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})). \tag{B.1}$$

To build intuition, consider a poor prediction of $\hat{y} = 0.2$ vs. a better prediction of $\hat{y} = 0.9$ for a positive example (*i.e.* $y = 1$). For the poor prediction, the value of the loss is computed as follows:

$$\mathcal{L} = -(1 \log(0.2) + 0 \log(0.8)) = -1 \log(0.2) \approx 0.70. \tag{B.2}$$

For the good prediction, the value of the loss is as follows:

$$\mathcal{L} = -(1 \log(0.9) + 0 \log(0.1)) = -1 \log(0.9) \approx 0.05. \tag{B.3}$$

Thus, this simple example, makes clear that the value of the cross-entropy loss function is higher for bad predictions and lower for good ones.

For a classification task with more than two possible label classes (*a.k.a.* multi-class classification), the cross-entropy loss function is the sum of individual cross-entropy loss terms for each label:

$$\mathcal{L} = \sum_{c=1}^{C} y_c \log(\hat{y}_c), \tag{B.4}$$

where $y_c \in \{0, 1\}$ refers to the value at index c of the one-hot vector $\boldsymbol{y} \in \{0, 1\}^C$ (*i.e.* it is either 0 or 1) and where $\hat{y}_c \in [0, 1]$ refers to the value at index c of the predicted output vector $\hat{\boldsymbol{y}}$ (*i.e.* it ranges from 0 to 1 and represents the predicted probability for class c).

Typically, a task is formulated to minimize the loss function when applied to all training examples (*i.e.* by summing up the loss values for each example). Thus, for an object classifier, the following loss would be used:

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}), \tag{B.5}$$

where i represents the index of a specific training datapoint.

B.1.3 Model

Deep learning refers to the use of *artificial neural networks* (*a.k.a.* deep neural networks), which comprise a class of algorithms[6] that is very loosely inspired by how the human brain processes information. Deep neural networks consist of multiple *layers* that successively process the input to the network, much like how the human successively processes raw visual input from the eyes, in order to produce an output. Some layers contain *parameters* (*a.k.a. weights*) that need to be optimized in order for the network to perform the given task well. In appendix B.2, we go into detail for the class of models discussed in this thesis: convolutional neural networks.

B.1.4 Training procedure

A deep neural network is typically trained by updating its parameters via *back-propagation* in order to minimize a loss function.

B.1.4.1 Backpropagation

Backpropagation refers to a class of algorithms in which the gradients $\frac{\partial \mathcal{L}}{\partial \theta}$ of a loss function with respect to a network's parameters, θ, are computed efficiently[7] for a single input-output pair (*i.e.* $(\boldsymbol{x}, \boldsymbol{y})$). The gradient captures how much each parameter should change (*i.e.* magnitude) and in which direction (*i.e.* positive or negative) in order to increase the loss value for that particular input-output pair.

B.1.4.2 Gradient descent

Because we are interested in decreasing (*i.e.* minimizing) the loss, we typically update parameters by taking a step in the negative direction of the gradient; this is known as *gradient descent* and is given by the following *update rule*:[8]

$$\theta \coloneqq \theta - \gamma \frac{\partial \mathcal{L}}{\partial \theta}, \tag{B.6}$$

where γ is the step size (*a.k.a. learning rate*).

[6] An *algorithm* is a well-defined process to perform a computation or specific task. It is analogous to a cooking recipe that instructs a novice chef how to make a scrumptious scone.

[7] As opposed to naively computing the gradient for every network parameter independently, which would be computationally expensive.

[8] $A \coloneqq B$ denotes *assigning* the value of A to B in computer science notation. In the case of eq. (B.6), the new value of θ is equal to to right hand side of the equation.

In practice, *stochastic gradient descent* (SGD), an iterative method for updating parameters based on random (*i.e.* stochastic) subsets (*a.k.a. batches*) of the training dataset,[9] is typically used:

Algorithm 1: Stochastic gradient descent (SGD)

Data: Training data $\mathcal{D}$
Hyperparameters: T (# training steps), B (batch size), γ (learning rate)
Randomly initialize network parameters θ
for $t = 1 \dots T$ **do**
  Randomly sample a batch: $\{\boldsymbol{x}_b, \boldsymbol{y}_b\}_{b=1}^{B} \sim \mathcal{D}$
  Compute loss for every item in the batch: $\{\mathcal{L}_b\}_{b=1}^{B}$
  Update θ using batch's gradient: $\theta \coloneqq \theta - \frac{\gamma}{B} \sum_{b=1}^{B} \frac{\partial \mathcal{L}_b}{\partial \theta}$
end

The optimization settings T (number of training steps), B (batch size), and γ (learning rate) are examples of **hyperparameters** for a network. A hyperparameter is a parameter that is typically set prior to training by a human; this is in contrast to the network's parameters, which are automatically learned.

B.2 Convolutional neural networks

Convolutional neural networks (CNNs) are a particular kind of deep neural networks that is most frequently used on *visual data* (*e.g.* an image). They are distinguished by their use of *convolutional layers*, which enable them to recognize distinctive visual patterns (*e.g.* furry ears) regardless of their location in an image. This property is known as *shift invariance*.[10]

In this section, we briefly survey the basic components of a CNN.

A *layer* refers to an operation that is applied to an input *tensor* and produces an output tensor, which may or may not be the same shape as the input tensor.

B.2.1 Linear layers

The basic building block of deep neural networks are *linear layers* that *linearly combine* (see appendix A.4.1) an input tensor with learned parameters (*i.e.* weights) to produce an output tensor.

There are two basic kinds of linear layers: fully-connected layers and convolutional layers.

[9] By taking random subsets, SGD approximates the actual gradient, which would need to be computed for the entire dataset.

[10] Here, *shift* refers to shifting the position of a pattern and *invariance* connotes remaining unchanged regardless of changes in another property (in this case, changes in spatial shift).

B.2.1.1 Fully-connected layer

A fully-connected layer is a layer in which a unique weight is used to "connect" every input neuron (*i.e.* element in the input tensor) to every output neuron (*i.e.* element in the output tensor).

2-layer example. Consider the following neural network $\Phi : \mathbb{R}^3 \to \mathbb{R}^2$ that contains two linear layers. Let the following matrices and vectors be the parameters of the first and second layer (superscript denotes the layer index) defined as follows:

$$\boldsymbol{W}^1 \in \mathbb{R}^{3\times 4}, \boldsymbol{W}^2 \in \mathbb{R}^{4\times 2}, \boldsymbol{b}^1 \in \mathbb{R}^3, \boldsymbol{b}^2 \in \mathbb{R}^2, \tag{B.7}$$

and let the following tensors be defined as the input, intermediate (*i.e.* activation), and output tensors respectively:

$$\boldsymbol{x} \in \mathbb{R}^3, \boldsymbol{z} \in \mathbb{R}^4, \hat{\boldsymbol{y}} \in \mathbb{R}^2. \tag{B.8}$$

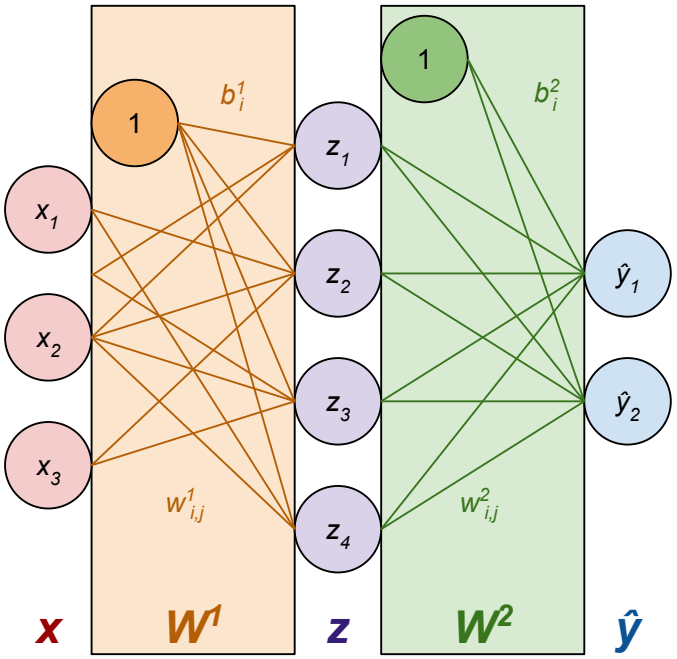


Figure B.1: Diagram of two fully-connected layers.

Then, layer 1 can be defined as the function $f^1 : \mathbb{R}^3 \to \mathbb{R}^4$, where the element at the j-th position in the output tensor $\boldsymbol{z}$ is given by

$$z_j = (\sum_{i=1}^{3} x_i \cdot w_{i,j}^1) + b_j^1. \tag{B.9}$$

Using matrix multiplication, we can write the function f^1 as follows:

$$\boldsymbol{z} = f^1(\boldsymbol{x}) = \boldsymbol{W}^{1^T}\boldsymbol{x} + \boldsymbol{b}^1, \tag{B.10}$$

where $\boldsymbol{W}^{1^T} : \mathbb{R}^4 \to \mathbb{R}^3$ is the transposed matrix of $\boldsymbol{W}^1$ (*i.e.* matrix with its rows and columns swapped). Layer 2 can be similarly defined as the function $f^2 : \mathbb{R}^4 \to \mathbb{R}^2$, with the j-th position element in its output given by

$$\hat{y}_j = (\sum_{i=1}^{4} z_i \cdot w_{i,j}^2) + b_j^2, \tag{B.11}$$

and re-written using matrix multiplication as

$$\hat{\boldsymbol{y}} = f^2(\boldsymbol{z}) = \boldsymbol{W}^2\boldsymbol{z} + \boldsymbol{b}^2. \tag{B.12}$$

Thus, a fully-connected layer linearly combines an input tensor with learned parameters: a weights tensor (*i.e.* $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$) and a bias tensor (*i.e.* $\boldsymbol{b}^1$ and $\boldsymbol{b}^2$).

As in this example, the input and output tensors of a fully-connected layer are most commonly vectors (*i.e.* 1st-order tensors); thus it follows that the weights tensor is a matrix (*a.k.a.* weights matrix) and the bias tensor is a vector (*a.k.a.* bias vector or bias term).

General form. Now, we can write a general function $f_{\text{fc}} : \mathbb{R}^M \to \mathbb{R}^N$ with weight matrix $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ and bias term $\boldsymbol{b} \in \mathbb{R}^N$ to describe a fully-connected layer that takes as input M-D vectors and outputs N-D vectors:

$$f_{\text{fc}}(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}. \tag{B.13}$$

By definition, the j-th element of the output tensor $\boldsymbol{z} = f_{\text{fc}}(x)$ is given by

$$z_j = (\sum_{i=1}^{M} x_i \cdot w_{i,j}) + b_j. \tag{B.14}$$

Working out an example. Finally, let us work out an example. Let us define the weight matrices in fig. B.1 as follows:

$$\boldsymbol{W}^1 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}, \boldsymbol{W}^2 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}, \tag{B.15}$$

and let both bias vectors be filled with 1. Then, the transposed weight matrices are as follows (it's easier to reason with the transposed matrices):

$$\boldsymbol{W}^{1^T} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}, \boldsymbol{W}^{2^T} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}. \tag{B.16}$$

Now, consider the input vector $\boldsymbol{x} = \begin{bmatrix} -2 & 0 & 1 \end{bmatrix}$.

Using the above equations, we compute the intermediate tensor $\boldsymbol{z}$ to be as follows:

$$\boldsymbol{z} = \begin{bmatrix} -2 \cdot 0 + 0 \cdot 1 + 1 \cdot 2 \\ -2 \cdot 1 + 0 \cdot 2 + 1 \cdot 3 \\ -2 \cdot 2 + 0 \cdot 3 + 1 \cdot 4 \\ -2 \cdot 3 + 0 \cdot 4 + 1 \cdot 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -1 \end{bmatrix}, \tag{B.17}$$

and the output tensor $\hat{\boldsymbol{y}}$ to be as follows:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} 2 \cdot 0 + 1 \cdot 1 + 0 \cdot 2 + -1 \cdot 3 \\ 2 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 + -1 \cdot 4 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}. \tag{B.18}$$

B.2.1.2 Convolutional layers

In contrast to a fully-connected layer, which applies a unique weight to every input neuron, a **convolutional layer** applies the same set of weights (*a.k.a. convolutional filters* or *convolutional kernels*) to "neighborhoods" of input neurons. This property of sharing weights makes convolutional layers well-suited for handling visual information, as each filter can be tuned to recognize a particular pattern (*i.e.* oriented edges, cat ear).

1D convolution. To build intuition, let us consider a simple example of a 1D convolution.[11]

In this example, the same weights vector $\boldsymbol{w}$ is applied to small neighborhoods comprised of 3 input neurons each (*i.e.* (x_1, x_2, x_3), (x_2, x_3, x_4), and (x_3, x_4, x_5)) in a "sliding window" fashion. To compute the value of an output neuron (*i.e.* z_2), for every input neuron connected to it, multiply its value with the value of its connecting weight (*i.e.* given by the line color), and sum up the products as follows:

$$z_i = x_i \cdot w_1 + x_{i+1} \cdot w_2 + x_{i+2} \cdot w_3. \tag{B.19}$$

[11]Here, the dimensionality refers to the number of ways the weight moves. In this case, it can only move along one direction (*i.e.* left-to-right). A 2-D convolution is used for images and can move along 2 dimensions (*i.e.* up-and-down and left-to-right).

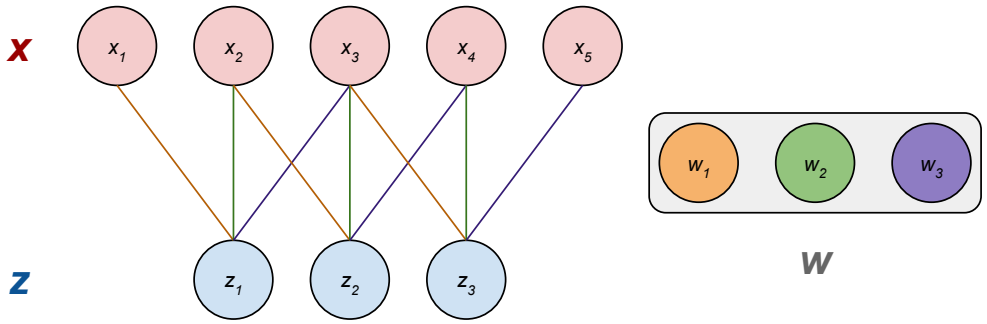


Figure B.2: 1D convolution. Notice how weights are re-used (i.e., line colors are repeated) and applied in a "sliding window" fashion.

Now, let's work out an example. Suppose the input and weight vectors are given as follows:

$$\boldsymbol{x} = \begin{bmatrix} 1 & 6 & 5 & 4 & 1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, b = 0. \tag{B.20}$$

Then, the resulting output vector is

$$\boldsymbol{z} = \begin{bmatrix} 1 \cdot -1 + 6 \cdot 0 + 5 \cdot 1 \\ 6 \cdot -1 + 5 \cdot 0 + 4 \cdot 1 \\ 5 \cdot -1 + 4 \cdot 0 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ -4 \end{bmatrix}. \tag{B.21}$$

This particular convolutional filter detects the presence and orientation of 1D "edges" (*i.e.* change in value within a neighborhood of neurons): A positive output neuron denotes an input neighborhood where the leftmost input neuron is smaller than the rightmost input neuron (*i.e.* value increases when comparing the left neuron with the right neuron), a negative output neuron denotes the reverse (*e.g.* value decreases), and the magnitude of the output neuron denotes the magnitude of the difference in values between the two outer input neurons.

2D convolution. For images, a 2D convolution is typically used, as images are two-dimensional. Similarly, to compute a value in the output tensor, consider the dot product between a convolutional filter (*i.e.* $\boldsymbol{w} \in \mathbb{R}^{3 \times H_k \times W_k}$) and a same-sized neighborhood in the input tensor.

In this example, we show one filter and the resulting output tensor slice (*i.e.* 3D-tensor with a channel depth of 1). Typically, a convolutional layer applies more than one filter. Because every filter produces an output tensor slice, the number of channels in the output tensor is equal to the number of filters used.

Thus, a 2D convolutional layer can be defined as $f_{\text{2DConv}} : \mathbb{R}^{C_i \times H_i \times W_i} \to \mathbb{R}^{C_o \times H_o \times W_o}$ with weights tensor $\boldsymbol{W} \in \mathbb{R}^{C_o \times C_i \times H_k \times W_k}$ and, if used, bias tensor $\boldsymbol{b} = \mathbb{R}^{C_o}$.[12]

[12]For conciseness and clarity, we do not use a bias tensor in the convolution examples.

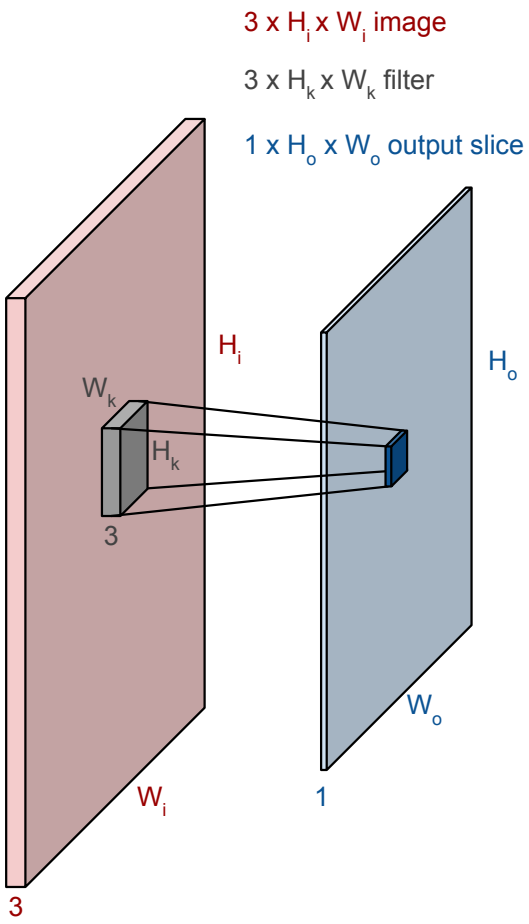


Figure B.3: 2D convolution.

Then, given a neighborhood in the input tensor that has the same spatial dimensions as the filters, *i.e.* $\boldsymbol{x}' \in \mathbb{R}^{C_i \times H_k \times W_k}$, its corresponding output value can be computed as follows:

$$z_k = (\sum_{c=1}^{C_i} \sum_{i=1}^{H_k} \sum_{j=1}^{W_k} w_{k,c,i,j} \boldsymbol{x}'_{k,c,i,j}) + b_k, \tag{B.22}$$

where $z_k \in \mathbb{R}$ is the output value that corresponds to applying the k-th filter to the neighborhood $\boldsymbol{x}'$

In addition to filter size and the number of filters, there are a few other hyperparameters for convolutional layers. *Stride* refers to the step size with which to "slide" the filter across the input tensor. *Padding* refers to adding additional input values (*i.e.* "padding" the input). The most common value with which to pad an input tensor is 0 and is known as *zero-padding*. In the examples shown here, we set the stride to be 1 (*i.e.* we shift the filter by an increment of 1) and use no padding. In the following examples, we modified the earlier 1D example

(fig. B.2) first to use zero-padding with a padding width of 1 (*i.e.* padding with 1 extra value on each side) and second to use a stride of 1.

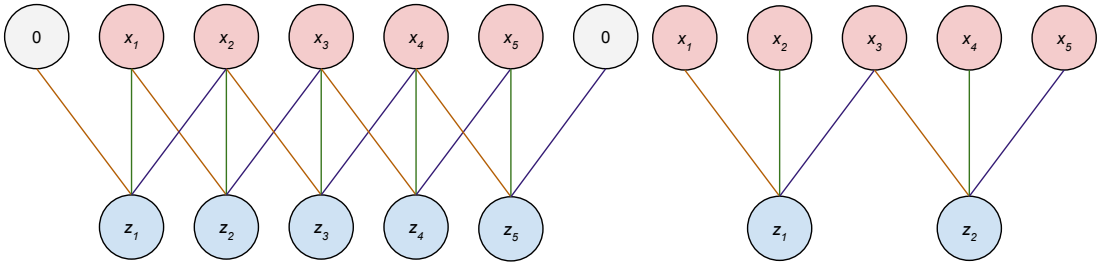


Figure B.4: More 1D convolution examples. Left: An example with padding width of 1 on each side. **Right:** An example with a stride of 2.

Now, we can write a formula for calculating the spatial size of the output tensor. To calculate the length of one output dimension O, we can use the following:

$$O = \frac{I - K + 2P}{S} + 1, \tag{B.23}$$

where I is the input size, K is the length of the filter, S is the stride step size, and P is the padding width, all along the same corresponding dimension. If the result is a fraction, round up.

B.2.2 Other layers

In addition to layers that linearly combine tensors with learned weights, there are several kinds of layers that enable a CNN to filter information such that only relevant features are propagated.

B.2.2.1 Activation layers

An activation layer typically applies a *non-linear*,[13] scalar function $g : \mathbb{R} \to \mathbb{R}$ to every element in the input tensor, thereby outputting a tensor of the same size.

The most common activation function used in modern CNNs is the **ReLU** function, which stands for *rectified linear unit*, and is defined as follows:

$$g(x) = \max(x, 0). \tag{B.24}$$

In practice, the ReLU function allows a CNN to disgard non-relevant information by setting all negative elements in an input tensor to 0. Because activation layers

[13] A linear relationship is one in which changes in the output are directly proportional to changes in the input. For instance, the function $f(x) = 2x$ is linear while the function $f(x) = \max(x, 0)$ is non-linear. This is because there are times when x varies (*i.e.* when $x < 0$) and the variation in the input is not reflected in the output.

typically immediately follow linear layers, a CNN can adapt filters in the preceding linear layer to fire positively when they recognize relevant features. Then, the following ReLU layer could "forget" neurons that captured irrelevant features.

Now, we can take a look at a simple example. Given an input tensor

$$\boldsymbol{x} = \begin{bmatrix} -2 & 1 & 2 \\ 0 & -1 & 1 \\ 3 & -1 & 2 \end{bmatrix}, \tag{B.25}$$

the result of passing it through a ReLU layer would be as follows:

$$\boldsymbol{z} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 3 & 0 & 2 \end{bmatrix}. \tag{B.26}$$

B.2.2.2 Pooling layers

A pooling layer forces information to be compressed spatially. Similar to 2D convolutions, which operate on spatial neighborhoods of elements in an input tensor, most pooling layers operate on neighborhoods in the input.

The **max pooling** operation chooses the maximum value from a set of inputs (*i.e.* spatial neighorhood) as the output value, while **average pooling** sets the output value as the average (*i.e.* mean) of a set of input values.

Because pooling layers are typically used to compress information *spatially*, they usually leave the channel dimension unchanged and apply pooling functions (*i.e.* max or average pooling) to spatial neighborhoods of tensor slices, that is, they are applied to every channel independently.

Now, we can define the pooling functions precisely. Given $\boldsymbol{x}' \in \mathbb{R}^{H_k \times W_k}$, a tensor slice representing a spatial neighborhood, max pooling and average pooling are defined as

$$g_{\text{maxpool}}(\boldsymbol{x}') = \max_{i=1}^{H_k} \max_{j=1}^{W_k} x'_{i,j} \text{ and } g_{\text{avgpool}}(\boldsymbol{x}') = \frac{1}{H_k \cdot W_k} \sum_{i=1}^{H_k} \sum_{j=1}^{W_k} x'_{i,j}. \tag{B.27}$$

Finally, let's work through a simple example.

Given an input tensor slice as follows:[14]

$$\boldsymbol{x} = \begin{bmatrix} -2 & 1 & -3 & 4 \\ 3 & -2 & 0 & -1 \\ 2 & 1 & -1 & -2 \\ 0 & -2 & 1 & -3 \end{bmatrix}. \tag{B.28}$$

[14]For simplicity, we are considering an input tensor with only one channel. For tensors with more than one channel, the same procedure would be applied to every channel in the input tensor.

The result of passing it through a max pooling layer that uses a 2×2 neighborhood is as follows:

$$\boldsymbol{z} = \begin{bmatrix} 3 & 1 & 4 \\ 3 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix}. \tag{B.29}$$

B.2.2.3 Regularization layers

There are also a number of layers that regularize the activation tensor (*e.g.* via dropout, batch normalization, etc.). For brevity, we only describe dropout.

A **dropout layer** randomly "drops" (*i.e.* sets to 0) input values and can be described as applying the following function $g : \mathbb{R} \rightarrow \mathbb{R}$ element-wise (*i.e.* to every input value):

$$g(x) = \begin{cases} x & p \leq 0.5, \\ 0 & \text{otherwise}, \end{cases} \tag{B.30}$$

where p is a unique random number generated afresh every time the function is applied. The result is that approximately half of the activation tensor is "dropped". This simple technique forces the model not to be too dependent on any one feature, as it will be dropped half of the time. This tends to improve a model's robustness (*i.e.* ability to perform well under a variety of conditions) and overall performance.

B.2.3 Putting it all together

Now that we've discussed various components of a CNN and aspects of a deep learning set-up, let's put this knowledge all together.

B.2.3.1 Model architecture

A CNN **architecture** refers to the specific configuration of layers and their settings (*i.e.* filter size, number of output channels, stride, padding).

In fig. B.5, we show a diagram detailing the AlexNet (Krizhevsky et al., 2012) architecture, which was one of the earliest demonstrations of the power of CNNs on object classification. This model uses around 61 million parameters (*i.e.* total number of scalars in all weight and bias tensors). Because a CNN is typically highly-parameterized, it is not feasible to understand a model simply by examining its parameters, due to the sheer volume of them.

Other popular CNN architectures include VGG networks (*e.g.* VGG16) (Simonyan et al., 2015), GoogLeNet (Szegedy et al., 2015), and residual networks (*e.g.* ResNet50) (He et al., 2016).

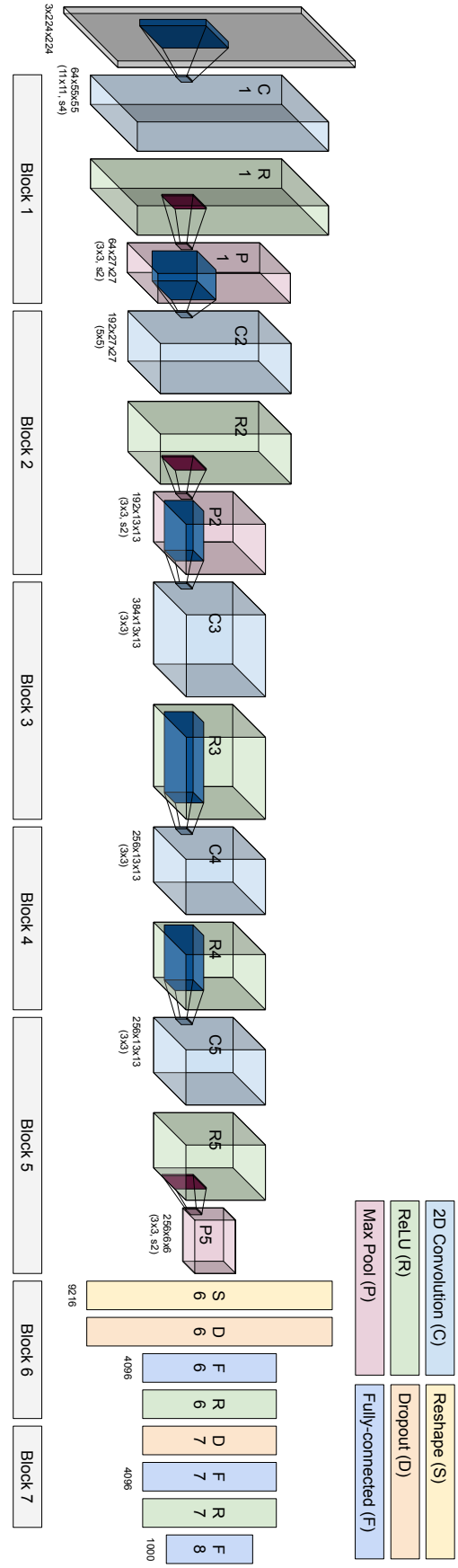


Figure B.5: AlexNet architecture.

C

Other interpretability papers

Contents

In this appendix, we include two first-author papers (Fong et al., 2019c; Rebuffi et al., 2020) that are related to the subject of this thesis.

C.1 Occlusions for Effective Data Augmentation in Image Classification

The following paper was presented at the workshop on Interpreting and Explaining Visual Artificial Intelligence Models, which was co-located at the IEEE International Conference of Computer Vision (ICCV) at Seoul, South Korea in 2019 (Fong et al., 2019c).

In Fong et al., 2019c, we demonstrate how to improve the performance of object classifiers by leveraging input-level perturbations as a data preprocessing step.

Occlusions for Effective Data Augmentation in Image Classification

Ruth C. Fong*
University of Oxford

Andrea Vedaldi
Facebook AI Research

Abstract

Deep networks for visual recognition are known to leverage "easy to recognise" portions of objects such as faces and distinctive texture patterns. The lack of a holistic understanding of objects may increase fragility and overfitting. In recent years, several papers have proposed to address this issue by means of occlusions as a form of data augmentation. However, successes have been limited to tasks such as weak localization and model interpretation, but no benefit was demonstrated on image classification on large-scale datasets. In this paper, we show that, by using a simple technique based on batch augmentation, occlusions as data augmentation can result in better performance on ImageNet for high-capacity models (e.g., ResNet50). We also show that varying amounts of occlusions used during training can be used to study the robustness of different neural network architectures.

1. Introduction

Robustness to occlusions is an important property of image recognition systems. That is, a robust image classifier should be able to solve the problem even if only a portion of the object of interest is visible in an image. However, the image classification datasets commonly used to train high-performance models such as deep neural networks are strongly affected by the so called "photographer bias". Among other things, this bias means that the main subject of these pictures tends to be centred and clearly visible. As a consequence, learning a model on such data may results in "lazy" networks that focus too much on easily recognizably details (such as the face of a cat) and cannot understanding other, more subtle cues (such as the cat's body) that may be important in harder scenarios.

A few authors have proposed to address this issue by augmenting the training data via simulated occlusions. While details change depending on the specific method, the general idea is that, if part of the image is not visible at training time, then the network should be stimulated to learn to recognize all available evidence, thus avoiding to over-rely on the most obvious evidence. However, the success of these techniques has been mixed. [21, 16] showed improvements on the ability of the network to localize objects but not on the original task of object recognition. [1] demonstrated better performance in classification performance in simpler datasets such as CIFAR10 [6] but not in larger, more complex ones such as ImageNet [13] (as confirmed in our experiments and in [3]).

A hypothesis for this behaviour is that training using occlusion augmentation improves the robustness of the model to occlusions, but that this does not correspond to a test-time performance improvement because the test set does not, in fact, contain occlusions.

In this paper, we show that this is not the case. The issue can be solved, and a performance improvement observed consistently, provided that the augmentation is incorporated properly in the training procedure. We make three main contributions: (1) We demonstrate that augmenting image batches with several versions of the same image, in the spirit of *batch augmentation* [5], allows occlusion augmentation to consistently outperform the baselines on for CNN architectures that are sufficiently powerful (e.g., ResNet50). (2) We present a detailed analysis of why occlusion augmentation has not yielded improvements in the past. (3) We conduct a thorough investigation on how to optimally tune occlusion augmentation, showing differences as a function of the model architecture. For example, we demonstrate that more powerful models (e.g., ResNet50 [4]) can handle, and benefit from, significantly more substantial occlusions during training than weaker ones (e.g., AlexNet [7]).

2. Related work

Occlusions have been successfully used for model interpretability and weak localization. A few attribution methods have used fixed [22], stochastic [11], and optimized [2] occlusions to diagnose "where" a network is "looking" in the input for evidence for its prediction. A few works have demonstrated that applying random [16] or optimized [21] occlusions to the input or intermediate activations [20] can improve weakly supervised localization (but not necessarily image classification) by forcing a classification network to be robust to occlusions and thus rely other parts of an object besides its most discriminative parts.

*Work done as a contractor at FAIR.

Cutout [1] and Hide-and-Seek [16] both introduce stochastic input-level occlusions: Cutout "drops" (i.e., zeros out) randomly positioned squares (Figure 2), while Hide-and-Seek divides an image into a square grid and "drops" grid patches independently (Figure 1). Hide-and-Seek [16] highlights its improvements of weak localization at the expense of classification performance on ImageNet [13]. Although Cutout [1] improves performance on CIFAR10 and CIFAR100 [6], [3] reported that it did not improve classification performance on ImageNet.

Other regularization methods related to occlusions are techniques inspired by Dropout [17], which "drop" parts of intermediate activation tensors, such as DropPath [8], Scheduled DropPath [24], Spatial Dropout [19] and DropBlock [3]. Whereas Dropout [17] drops a single voxel from a 3D activation tensor of a given input, DropPath [8] drops a whole branch of a network while Spatial Dropout [19] drops a whole slice in a 3D activation tensor associated to a filter. DropBlock [3] can be viewed as an extension of Cutout [1] applied to intermediate activations. In this method, contiguous blocks in each activation slice associated to a filter are dropped. These techniques, particularly DropBlock [3], yield modest but consistent improvements in ImageNet [13] classification performance; however, they all require architectural change and, in the case of Scheduled DropPath [24] and DropBlock [3], requires using a training schedule specific for its modules. Label smoothing [18] is another related regularization technique, in which noise is added to the training labels.

Recently, batch augmentation [5] was introduced as a way to augment existing data augmentation techniques by including multiple copies of the same image (i.e. copying an original batch M times) and applying data augmentation to each of the copies. When coupled with Cutout [1], batch augmentation significantly improved performance on small datasets like CIFAR10 and CIFAR100 [6].

Similar to [21], which uses CAM (class activation maps) [23], we explore using the heatmaps produced by attribution methods to occlude images during training. We focus on the gradient-based saliency method introduced in [12], which is closely related to Grad-CAM [14] and the linear approximation [10] at a specific layer. [12] shows that their method, when used to aggressively occlude images during training outperforms other baselines, including Grad-CAM [14]. While [12] focused on dataset compression and willingly sacrificed on task performance, we are interested in using occlusions to improve task performance.

Our work most directly builds off of Cutout [1], Hide-and-Seek [16], the gradient-based saliency method in [12], and Batch Augmentation [5].

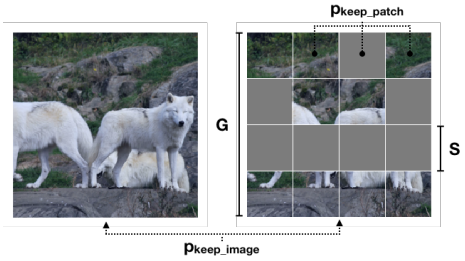


Figure 1. **Stochastic Hide-and-Seek [16] occlusions.** With probability $p_{\text{keep\_image}}$, the image is fully preserved (left). With $1 - p_{\text{keep\_image}}$, each cell in a disjoint $G \times G$ grid (with side length S) is randomly occluded with probability $1 - p_{\text{keep\_patch}}$ (right). For joint training, an image duplicated into two copies, where one is always occluded ($p_{\text{keep\_image}} = 0$) and the other never occluded ($p_{\text{keep\_image}} = 1$). White grid lines are used for illustration.

Figure 2. **Stochastic Cutout [1] occlusions.** For a given image, the center points of N square occlusions of side length S are independently and randomly placed ($N = 6, S = 56$ in these examples).

3. Method

We introduce a simple paradigm for using occlusions effectively as data augmentation. For every image $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$, we generate a pattern of occlusion $\boldsymbol{m} \in \mathbb{R}^{1\times H\times W}$ using one of the methods described below. Then, for a given batch of images $\boldsymbol{X} \in \mathbb{R}^{B\times 3\times H\times W}$, we copy the batch. We apply the set of occlusions, $\boldsymbol{M} \in \mathbb{R}^{B\times 1\times H\times W}$, to one copy of the batch, leaving the other batch unoccluded, and *train jointly* with one combined batch: $(\boldsymbol{X}, \boldsymbol{X} \odot \boldsymbol{M})$. We occlude a pixel by replacing it with the mean average colour (i.e. setting it to zero after mean normalization). Our joint training is inspired by batch augmentation [5].

Stochastic occlusions. We first consider two existing ways of generating occlusions stochastically: Hide-and-Seek [16] and Cutout [1]. *Hide-and-Seek (H&S)* divides an image into a $G \times G$ grid and drops patches in the grid independently with probability $1 - p_{\text{keep\_patch}}$, where $p_{\text{keep\_patch}} \in [0, 1]$ denotes the probability of preserving the original patch (Figure 1). *Cutout (CO)* drops N square patches[1] of side length S; the center of these patches are placed uniformly at random on the whole image, thereby allowing for some patches to "overflow" off the image, as

[1] The original Cutout paper [1] only considers $N = 1$ patch.

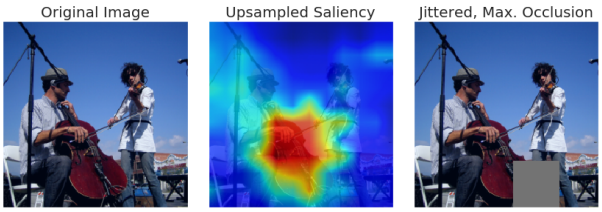


Figure 3. **Saliency-based [12] occlusions.** A saliency map at a given layer is generated according to Equation (1) and is then bilinearly upsampled to image size (middle). Next, the $S \times S$ maximal patch is extracted and jittered by τ (right). Here, a "cello" saliency map is generated at VGG16-BN's conv5, with $S = 56$ and $\tau \in [-16, 16]$.

done in [1] (Figure 2).

To analyse whether jointly training with an image occluded and unoccluded in the same batch is necessary, we introduce another hyperparameter, $p_{\text{keep\_image}} \in [0, 1]$. When using Hide-and-Seek occlusions without joint training (i.e., every image is in the batch exactly once), we show the full image with probability $p_{\text{keep\_image}}$. Otherwise, we show an image occluded with Hide-and-Seek-style dropout, in which a patch is preserved with probability $p_{\text{keep\_patch}}$. When $p_{\text{keep\_image}} = 0$, every image is potentially occluded; when $p_{\text{keep\_image}} = 1$, all images are unoccluded (i.e., standard training).

When comparing these two types of stochastic methods, Hide-and-Seek allows us to more easily and precisely define the amount of occlusion being applied on average. This is because Cutout occlusions are allowed to flow over image boundaries and can overlap with one another in the case of $N > 1$ patches being cut out. Pairing Hide-and-Seek with standard data augmentation (i.e., random cropping and resizing) simulates dynamic occlusions while its disjoint grid makes it easy to reason about the occlusions being applied. Nevertheless, Cutout is more comparable to the next type of occlusions we consider: saliency-based occlusions.

Saliency-based occlusions. We also consider generating occlusions based on saliency. Given a saliency heatmap, we extract an occlusion that is most salient compared to other potential patches. In this way, we use saliency heatmaps to guide occlusion locations as opposed to randomly sampling their locations. This allows us to fairly compare against Cutout [1] as we consider occlusions of the same size.

In our experiments, we use [12]'s gradient-based saliency method, which we summarize here (Figure 3; see [12] for more details). For a given layer l, a saliency heatmap $\boldsymbol{s} \in \mathbb{R}^{H_l \times W_l}$ can be generated by computing the Frobenius norm of the product of layer l's activation and gradient vectors, $\boldsymbol{x}'_{i,j}, \boldsymbol{g}_{i,j} \in \mathbb{R}^{K_l}$, at every spatial location (i, j):

$$s_{i,j} = \left\| \boldsymbol{g}_{i,j} \boldsymbol{x}'^{\top}_{i,j} \right\|_{\text{F}} = \|\boldsymbol{g}_{i,j}\| \cdot \|\boldsymbol{x}'_{i,j}\| \tag{1}$$

Intuitively, [12]'s saliency method precisely characterizes the contribution of every spatial location to the gradient of a hypothetical, subsequent 1×1 convolution weights tensor initialized with identity. We chose to use [12] because it generates high-quality, dense saliency maps at any network depth. In contrast, Grad-CAM [14] only works at the last conv layer.

For every image, we compute a saliency map with respect to the ground truth label and upsample the saliency map to the original image resolution $\mathbb{R}^{H \times W}$. We then find the square patch with side length S of the upsampled saliency map[2]. Finally, we add a small amount of jitter τ to the extracted patches. Unlike Cutout, we do not allow our patch to overflow the image boundaries (i.e., it will always be fully contained in the image).

4. Experiments

4.1. Implementation details

All models were trained for 100 epochs with the learning rate decayed by 0.1 every 30 epochs (i.e., at 30, 60, and 90 epochs). The initial learning rate for ResNet50 [4] and VGG16-BN was 0.1; for AlexNet [7] and VGG16 [15] it was 0.01[3]. All models used an original batch size of 256; jointly trained models used an actual batch size of 512, in which the original batch is duplicated and one copy is occluded. The actual batch was split across 8 GPUs. The original batch was preprocessed using standard data augmentation[4]: random cropping to 224×224, horizontal flipping, data normalization to $\mu = 0, \sigma = 1$.

When jointly training, the standard data augmentation (i.e., random cropping, etc.) occurs before the batch is duplicated, so the images are identical except for the regions that are occluded, i.e., $\boldsymbol{M}$. This differs from batch augmentation, in which images are preprocessed *independently* rather than *identically*.

Baselines. For non-joint training baselines, we trained networks in the usual fashion without occlusions. We introduced another set of baselines to account for the possible effect of doubling training time via joint training. The joint training baseline refers to networks that have been trained without occlusions but with duplicated batches, that is, every image appears exactly twice in the batch.

4.2. Stochastic occlusions

Experiment set-up. We first trained networks using Hide-and-Seek [16] occlusions. For these experiments, we divided

[2] In practice, we do this by convolving the saliency map with a $S \times S$ convolutional filter with stride T and filled with 1s.

[3] This was chosen for the non-batch normalization models based on grid search over the following learning rates: 0.1, 0.05, 0.01, 0.005, 0.001.

[4] We used default PyTorch ImageNet preprocessing: https://github.com/pytorch/examples/tree/master/imagenet

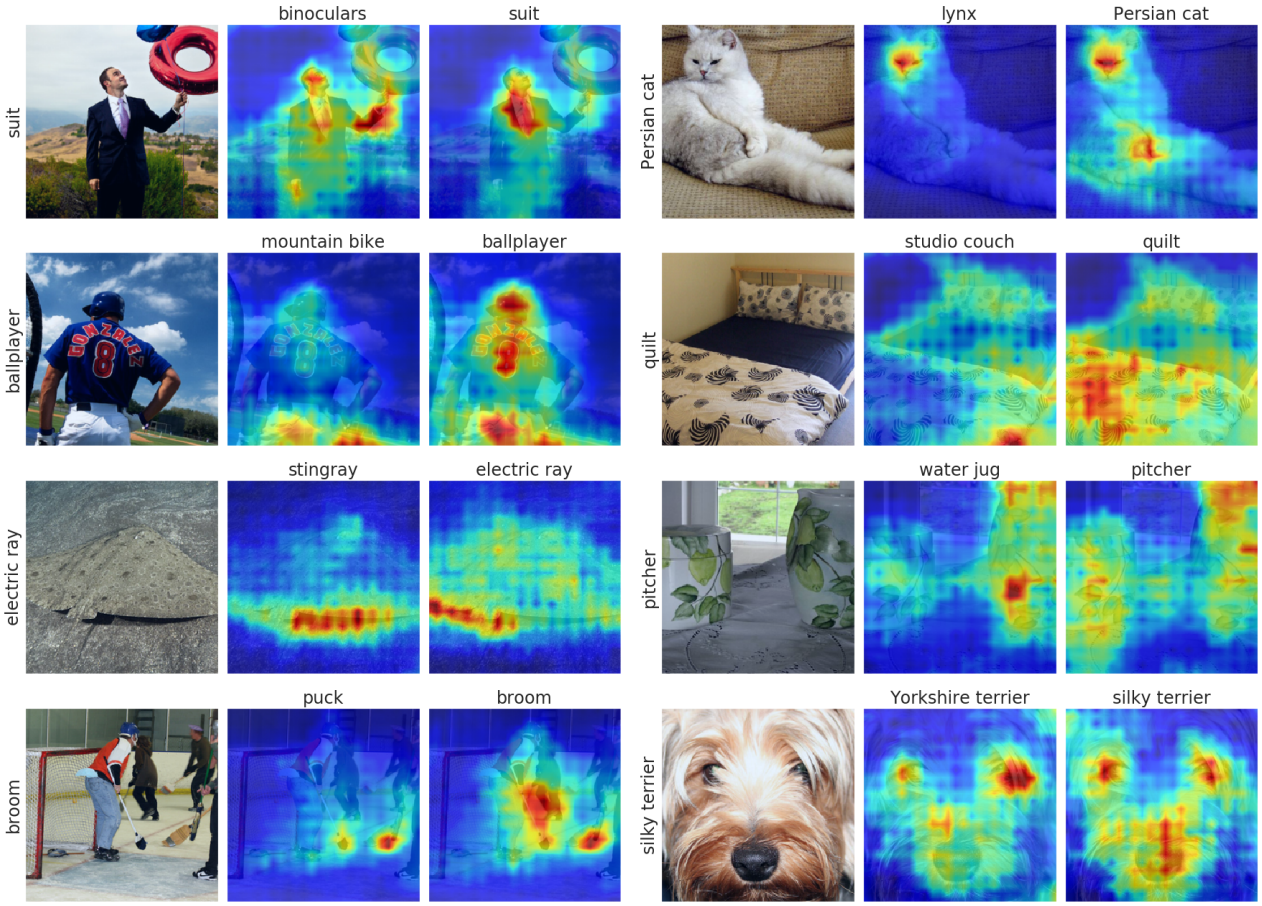


Figure 4. **Saliency visualizations [12] comparing standard vs. occlusion-augmented ResNet50s (layer3.0.conv1).** Left: original image; Middle: visualization for a ResNet50 baseline without joint training (76.43% [top-1] 93.17% [top-5]); Right: visualization for a ResNet50 trained jointly with Hide-and-Seek ($p_{\text{keep\_patch}} = 0.6$; 76.43% [top-1] and 93.17% [top-5]). Top predicted classes by the network are above; ground truth labels are on the left.

images into 4×4 grids ($G = 4$) and preserved patches with $p \in \{0.5, 0.6, \ldots, 0.9\}$. With a 224×224 cropped image size and grid size of $G = 4$, the size of the Hide-and-Seek patches were 56×56 ($S = 56$).

Based on our Hide-and-Seek results, we then train select networks (ResNet50 and AlexNet) using Cutout-style occlusions. Here, we occluded an image with either $N \in \{1, 2, 4, 6, 8\}$ square patches with side lengths $S \in \{56, 84, 112\}$.

For both kinds of occlusions, we trained networks jointly, that is, every batch was doubled and one copy was preserved as is (i.e., with full images) while occlusions were applied to the other copy. At evaluation time, no occlusions are applied.

Results. Table 1 reports ImageNet top-1 and top-5 accuracy for various networks when trained jointly with Hide-and-Seek occlusions, while Table 2 and Table 3 reports results for ResNet50 and AlexNet respectively when trained jointly with Cutout occlusions.

Table 1 shows that ResNet50 improves significantly ($+0.62\%$ in top-1 and $+0.35\%$ in top-5 for the optimal $p^*_{\text{kp}} = 0.5$) when jointly trained with H&S occlusions. Furthermore, ResNet50 consistently beats the baseline (10 of 10 results improve) regardless of the $p_{\text{keep\_patch}}$ hyperparameter. However, for all other networks, the best improvements are negligible: $0.07\%, 0.04\%, 0.02\%$ in top-1 and $0.07\%, -0.05\%, -0.07\%$ in top-5 for VGG16-BN, VGG16, and AlexNet respectively. Consistent with the results reported in [5], the difference between the joint and non-joint baselines in Table 1 appears roughly correlated with network performance, with ResNet50 having no difference and while the others demonstrate significant improvement with joint training: top-1 baselines improve by $0.00\%, 0.84\%, 0.62\%, 0.95\%$ for VGG16-BN, VGG16, and AlexNet respectively.

Thus, we focus our attention on ResNet50 for Cutout

	ResNet50		VGG16-BN		VGG16		AlexNet	
	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
baseline (w/o joint)	76.40	93.10	74.11	91.81	71.75	90.45	56.39	79.19
baseline (w/ joint)	76.40	93.03	74.95	92.31	72.37	**90.91**	57.34	**79.72**
$p_{\text{keep\_patch}} = 0.9$	76.58	93.19	74.66*	92.29*	72.20	90.73	**57.36**	79.65
$p_{\text{keep\_patch}} = 0.8$	77.97	93.31	74.82	92.34	72.31	90.75	56.98	79.48
$p_{\text{keep\_patch}} = 0.7$	76.90	93.33	74.97	92.32	72.37	90.86	56.97	79.48
$p_{\text{keep\_patch}} = 0.6$	77.01	**93.45**	74.85	92.30	**72.41**	90.81	56.97*	79.35*
$p_{\text{keep\_patch}} = 0.5$	**77.02**	**93.45**	**75.02**	**92.38**	72.22	90.69	56.59	79.11

Table 1. **Stochastic Hide-and-Seek [16] occlusions (joint training).** ImageNet top-1 and top-5 accuracies (%) are averaged over 3 runs except where * (denotes 2 runs); stddev mean $= 0.14$ with range $[0.02, 0.31]$. ResNet50 notably improves by 0.62% when jointly trained with H&S occlusions.

	S =56		S =84		S =112	
N	top-1	top-5	top-1	top-5	top-1	top-5
1	76.72	93.33	76.58	93.09	76.86	93.38
2	76.55	93.12	76.94	93.32	76.96	93.36
4	76.82	93.47	77.07	93.47	**77.19**	**93.46**
6	**77.17**	**93.54**	**77.25**	**93.48**	76.80	93.39
8	76.80	93.36	76.94	93.33	76.39	93.18

Table 2. **Cutout for ResNet50 (joint training).** Results reported on one run. S = side length of square patch; N = # of patches to cut out. For comparison, the joint baselines are 76.40% (top-1) and 93.03% (top-5) and the best joint Hide-and-Seek ($p^*_{\text{keep\_patch}} = 0.5$) results are 77.02% (top-1) and 93.45% (top-5) from Table 1.

experiments. Table 2 shows that Cutout with joint training on ResNet50 nearly always improves on the baseline (23 of 25 results improve), regardless of the size and number of patches occluded (S and N). The best result improves 0.85% for top-1 and 0.38% for top-5 over baselines, with the top-1 improvement being substantially higher with the best Cutout hyper parameters (77.25% with $N = 6, S = 84$) than that with the best Hide-and-Seek ones (77.02% with $p_{\text{keep\_patch}} = 0.5$).

In contrast, Table 3 shows that Cutout with joint training on AlexNet rarely improves on the joint baseline (only 1 of 25 results improves; we include this table for comparison with saliency-based occlusions in Section 4.4).

Taken together, these results suggest that, for complex datasets like ImageNet, a suitably powerful architecture like ResNet50 is likely necessary to benefit from occlusion augmentation.

Occlusions as a stethoscope for model capacity. The results for both kinds of stochastic occlusions (Table 1 and Table 2) peak in performance with the best hyper-parameters and then roughly monotonically decrease from that point. Thus, training with occlusions is beneficial from a model understanding perspective, as it provides a way to identify and quantify an architecture's upper bound for handling occlusions at evaluation time. For Hide-and-Seek (Table 1), we see that the optimal $p^*_{\text{keep\_patch}} \in [0.5, 0.6]$ for ResNet50 and VGG16-BN, $p^*_{\text{keep\_patch}} \in [0.6, 0.7]$ for VGG16, and $p^*_{\text{keep\_patch}} = 0.9$ for AlexNet. This suggests that AlexNet can only handle a small amount of occlusion (images occluded up to 10% on average), while VGG16-BN and ResNet50 are capable of handling images that have been occluded up to 50% on average, when trained properly with occlusions (ResNet50 and VGG16-BN may be able to handle more than 50%, but this was not tested).

Visualizations. Figure 4 compares a ResNet50 non-joint baseline against a ResNet trained jointly with Hide-and-Seek $p_{\text{keep\_patch}} = 0.6$ best by using [12]'s saliency method on layer3.0.conv1. Here, we visualize saliency maps for a few examples in which the occlusion-augmented network was correct and the baseline was wrong. Qualitatively, we observe difference in the models' predictions in their visualizations: In the suit image, the augmented network focuses on the tie while the baseline is attracted to the man's gaze and elbow. Same with the ball player, we see the baseline's mistake in focusing on the bottom edge of the image. In line with previous work [16, 21, 20], we also observe that visualizations of the augmented network tend to cover the object surface more than those of the baseline model.

4.3. Joint vs. non-joint training

We then thoroughly tested the necessity of joint training to make occlusion augmentation effective. We trained networks with Hide-and-Seek occlusions *without* joint training by introducing another hyper-parameter $p_{\text{keep\_image}}$ that determines whether an image is left completely unoccluded (see Section 3 for more details). We train these networks with $p_{\text{keep\_image}} = \{0.0, 0.1, \ldots, 1.0\}$ and $p_{\text{keep\_patch}} = \{0.5, 0.6, \ldots, 0.9\}$. We then compare those networks with our baselines and our jointly trained networks from Table 1. If joint training is not strictly necessary, we would expect

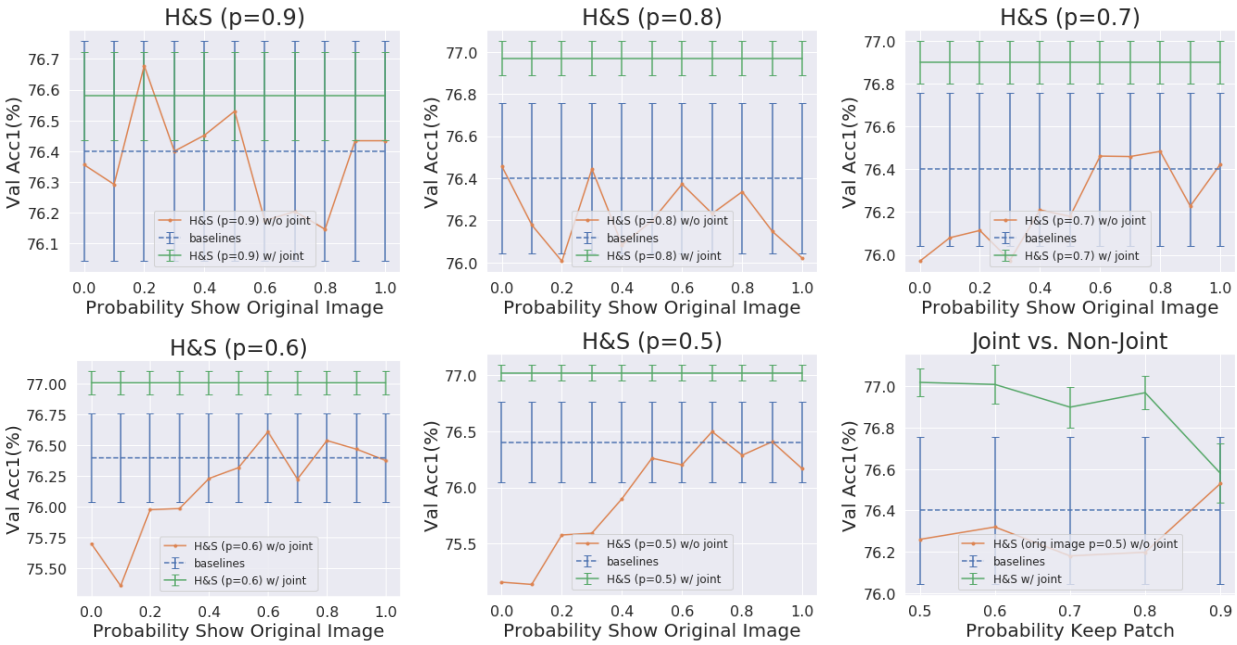


Figure 5. **Joint vs. Non-Joint Training on ResNet50.** We show that joint training (i.e., same image occluded and unoccluded in a mini-batch; red lines) is necessary to improve over baselines (dotted lines) compared to leaving an image unoccluded randomly ($p_{\text{keep\_image}}$). All plots except the bottom right one show ResNet50 baselines and Hide-and-Seek (H&S) joint and non-joint training for a given $p_{\text{keep\_patch}}$ as $p_{\text{keep\_image}}$ varies. The bottom right plot compares H&S joint and non-joint training for $p_{\text{keep\_image}} = 0.5$ as $p_{\text{keep\_patch}}$ varies.

our non-jointly trained networks to beat the baselines.

Figure 5 shows that this is not the case. Overwhelming, the non-jointly trained networks (green lines) perform worse than our baselines (dotted lines). While we might expect that when $p_{\text{keep\_patch}} = 0$, that is, when images are always occluded and thus the training domain might be too different from the test domain, it is surprising that even when showing full images half of the time ($p_{\text{keep\_patch}} = 0.5$), we do not see an improvement. This suggests that that seeing an image occluded and unoccluded *in the same batch* is necessary for occlusion augmentation to work well. Our finding are consistent with [3]'s observation that Cutout did not improve ImageNet classification performance.

We also briefly explored finetuning models on full images after they have been trained on exclusively occluded images but did not see an improvement over baselines.

4.4. Saliency-based occlusions

Experiment set-up. For AlexNet, VGG16, and VGG16-BN, we train networks with occlusions based on [12]'s saliency maps at the following layers (post-ReLU but pre-pooling): conv3, conv4, and conv5[5]. For ResNet50, we train networks on saliency maps on the max pool before the first block and on the very first convolutional layers in the first, second, and third blocks before batch normalization. Given

[5]For VGG16(-BN), convX refers to the last convolutional layer in the X-th block.

	S =56		S =84		S =112	
N	top-1	top-5	top-1	top-5	top-1	top-5
1	57.32	79.58	**57.24**	**79.65**	**57.32**	**79.66**
2	**57.49**	**79.67**	57.22	79.61	57.03	79.32
4	57.32*	79.57*	56.65	79.29	56.16	78.85
6	56.94	79.37	56.39	78.84	55.37	78.16
8	56.49	79.14	55.52	78.30	54.95	77.91

Table 3. **Cutout for AlexNet (joint training).** Averaged over 2 runs except where * (denotes 1 run); standard deviation mean $= 0.92$ with range $[0.00, 0.32]$.

	VGG16-BN		VGG16		AlexNet	
	top-1	top-5	top-1	top-5	top-1	top-5
w/o joint	74.11	91.81	71.75	90.45	56.39	79.19
w/ joint	74.95	92.31	72.37	**90.91**	57.34	79.72
conv3	75.01	**92.41**	**72.48**	90.86	**57.42**	79.71
conv4	74.96	92.40	72.27	90.87	57.38	**79.74**
conv5	**75.06**	92.39	72.38	**90.90**	57.33	79.70
Best from Tbl 1	75.02	92.38	72.41	90.86	57.36	79.65

Table 4. **Saliency-based [12] occlusions for VGG16-BN, VGG16, and AlexNet (joint training).** Averaged over 3 runs; stddev mean $= 0.07$ with range $[0.03, 0.16]$. Hyper-parameters $N = 1$ occlusion, $S = 56$ side length, $\tau = 16$ jitter are used.

ResNet50	top-1	top-5
w/o joint	76.40†	93.10†
w/ joint	76.40†	93.03†
maxpool	76.57	93.19
layer1.0.conv1	76.21	93.06
layer2.0.conv1	76.53	93.00
layer3.0.conv1	76.36	93.12
Best from Tbl 1	**77.02**†	**93.45**†

Table 5. **Saliency-based occlusions for ResNet50 (joint training).** Results reported for 1 run (†denotes averaged over 3 runs). These hyper-parameters were used: $N = 1, S = 56, \tau = 16$.

a saliency heatmap, we extract a 56×56 Cutout-like patch that covers the most salient part of the image. We then jitter the patch uniformly by $\tau \in [-16, 16]$ pixels.

Results. Table 4 and Table 5 show that the best results from training jointly with saliency-based occlusions for all networks except ResNet50 are consistently better (albeit by a small margin) than the best results from training jointly with stochastic Hide-and-Seek occlusions. Most notably, a much smaller amount of saliency-based occlusion is needed to yield the comparable improvements to Hide-and-Seek occlusions (i.e., for VGG16-BN, occluding 6% of an image using saliency is comparable to occluding 50% on average using Hide-and-Seek). This is likely due to the fact that the saliency-based occlusions should be covering the most "important" parts of an image. Our saliency-based $N = 1$ occlusion of side length $S = 56$ is roughly comparable to Hide-and-Seek with a 4×4 grid ($G = 4, S = 56$) and $p_{\text{keep\_patch}} = 15/16 = 0.94$, that is, on average only one 56×56 patch is occluded. It is also is directly comparable with Cutout with the same hyper-parameters ($N = 1, S = 56$; see Table 2 and Table 3 for Cutout on ResNet50 and AlexNet respectively).

The slim differences between results from different layers suggests that occlusions based on [12]'s saliency method are reasonably robust to layer choice. Saliency-based occlusion also yields a lower mean standard deviation of 0.07 compared to 0.14 for Hide-and-Seek occlusions, due to the significantly less stochastic nature of saliency-based occlusion augmentation.

One limitation of our current approach is that we can extract one maximal patch, thereby limiting to a certain degree the size of our occlusions, which would need to be larger in order to match the effects of the best parameterizations of the stochastic methods. This limitation is likely the reason that results from saliency-based $N = 1$ occlusions on ResNet50 do not beat the best stochastic occlusion results, since a larger amount of occlusion is needed for Hide-and-Seek ($p_{\text{keep\_patch}} = 0.5$) and Cutout ($N = 6$ for $S = 56$).

4.5. Comparison with other regularization methods

Experimental set-up. We compare our method with variants of Dropout [3] and primarily follow [3]'s protocol (see Section 2 for more details). For Dropout [17], Spatial Dropout [19], and DropBlock [3], we follow [3]'s procedure and add dropout modules after every convolutional layer in the third and fourth block of ResNet50. For DropBlock, we also add its module to the skip connections in those blocks. For Dropout and Spatial Dropout, we train ResNet50 networks without joint training using $p_{\text{keep\_prob}} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, while for DropBlock, we use $p_{\text{keep\_prob}} \in \{0.75, 0.80, 0.85, 0.90, 0.95\}$ ($p_{\text{keep\_prob}}$ is analogous to $p_{\text{keep\_patch}}$). We also compare against label smoothing [18] with fixed $p = 0.1$.

We deviate from [3] in that we train for 100 epochs using a 30–60–90 epoch lr decay schedule (vs. their 300 epochs using a 100–200–265 schedule) to compare fairly with our method. We do not use a schedule to ease in the amount of dropout for DropBlock, as [3] reported that DropBlock without scheduling still yielded significant boosts over their ResNet50 baseline. We expected that these two changes would decrease the improvements observed in [3] but that those improvements would still persist.

Results. Table 6 shows that all the variants of Dropout methods under-performed our ResNet50 non-joint training baseline, suggesting that they are sensitive to and require the custom longer training schedule used in [3] in order to be effective (see [3] for results with the longer training schedule). Label smoothing also under-performed our occlusion augmentation training.

4.6. Comparison with Batch Augmentation [5]

Lastly, we compare our joint training paradigm with batch augmentation [5]. The key difference between batch augmentation and joint training is that, for joint training, all standard pre-processing occurs *before* image duplication; in contrast, for batch augmentation, pre-processing occurs *after* duplication. Thus, transformation from pre-processing are *identical* in joint training but independent (i.e., different) in batch augmentation. In all our previous results, we used joint training ($M = 2$ copies).

Batch augment > joint training. Table 7 shows results when we use batch augmentation to include stochastic Cutout occlusions during training, with fixed CO hyper-parameters $N = 6, S = 56$. The results for $M = 2$ in Table 7 improve upon and are comparable to our joint training Cutout results for $N = 6, S = 56$ in Table 2: 77.50% (top-1) and 93.63% (top-5) for batch augmented Cutout ($p_{\text{keep\_image}} = 0.5$[6])

[6] $p_{\text{keep\_image}}$ denotes the probability that an image copy is left unoccluded.

ResNet50	top-1 (%) non-joint	top-1 (%) joint	top-5 (%) non-joint	top-5 (%) joint
baseline from Table 1	76.40	76.40	93.10	93.03
Dropout [17] ($p_{\text{keep\_prob}} = 0.9$)	76.34	76.41	93.02	93.10
Spatial Dropout [19] ($p_{\text{keep\_prob}} = 0.9$)	75.95	76.31	92.77	93.04
DropBlock [3] ($p_{\text{keep\_prob}} = 0.95$ & $p_{\text{keep\_prob}} = 0.90^{\dagger}$)	75.88	76.33	92.77	92.98
Label smoothing [18] (0.1)	**76.64**	76.26	**93.25**	93.11
Best H&S ($p^{*}_{\text{keep\_patch}} = 0.5$) from Table 1 (ours)	–	77.02	–	93.45
Best CO ($N* = 6; S* = 84$) from Table 2 (ours)	–	**77.25**	–	**93.48**

Table 6. **Comparison with other regularization methods.** For Dropout variants, the best results from a search over several $p_{\text{keep\_prob}}$ values is reported. †For DropBlock, $p_{\text{keep\_prob}} = 0.95$ was the best for non-joint and $p = 0.90$ was best for joint.

	JT bsl (Tbl 1)		JT CO (Tbl 2)		BA bs		BA CO ($p_{\text{ki}} = 0.0$)		BA CO ($p_{\text{ki}} = 0.5$)	
M	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
2	76.40†	93.03†	77.17*	93.54*	77.27*	93.47*	**77.50***	**93.63***	**77.50**	93.61
4	–	–	–	–	77.71*	**93.79***	77.75	93.68	**77.82**	93.74

Table 7. **Batch augment (BA) vs. joint training (JT) for Cutout (CO) on ResNet50.** Averaged over 2 runs except where * (1 run) and †(3 runs); stddev mean $= 0.09$ and range $[0.02, 0.18]$ for 2-run results. Best results *per row* are in bold. CO hyper-parameters were $S = 56$ and $N = 6$. $M =$ # copies of an image in a mini-batch. $p_{\text{ki}} = p_{\text{keep\_image}}$.

	Batch Augment		Dataset Augment		Joint Training	
M	top-1	top-5	top-1	top-5	top-1	top-5
2	**77.27**	**93.47**	76.51	93.12	76.39	93.14
4	**77.71**	**93.79**	76.01	92.53	76.30	93.07

Table 8. **Baseline comparisons with different kinds of augmentation.** Results are from 1 run. $M =$ number of copies of an image in a mini-batch for BA and JT and in one epoch of training for DA.

vs. 77.17% (top-1) and 93.54% (top-5) for joint training. However, batch augmentation also significantly improves its respective baseline; thus, relative improvement of batch-augmented Cutout are smaller when compared to that of jointly trained Cutout: For $M = 2$, is quite slim for top-1 (and non-existent for top-5) when using $M = 4$ copies.

No full images needed. Most notably, Cutout with $p_{\text{keep\_image}} = 0.0$ achieves similar performance to that with $p_{\text{keep\_image}} = 0.5$. This suggests that one can train a network with images that are *always occluded* (i.e., without ever seen a full, natural image) and achieve superior inference-time performance on full images than standard training methods.

Baseline comparisons. Table 8 shows results when training baseline ResNet50 models with batch augmentation, dataset augmentation, and joint training. Dataset augmentation iterates through the training set M times. The only difference between batch and dataset augmentation is that batch augmentation ensures that all the copies of one image are in the *same* mini-batch; in dataset augmentation, the copies of one image are in *distinct* mini-batches. These results verify that of [5] in showing the necessity of having image in the *same* mini-batch to achieve superior performance.

5. Conclusion

We showed how to leverage occlusions as effective data augmentation to improve ImageNet classification performance. The primary insight from our work is using some variant of batch augmentation [5] (such as our joint training with an occluded and unoccluded copy of an image in the same batch) is necessary to gain this improvement. This suggests that further research on what is being learned during joint training and more broadly batch augmentation [5] is warranted. We also demonstrate training-time occlusions can be a way to understand model's upper bound for robustness to occlusions generally. There is likely room to improve our work here, particularly in exploring further the potential of batch augmentation [5], in developing better saliency-based approaches to occlusion augmentation, and in elucidating further the interaction between and impact of dataset and model complexity for effective occlusion augmentation. Further research could also be done on other kinds of occlusions, such as blur or random noise or even ignoring regions [9]. In conclusion, in contrast to other regularization techniques that require architectural changes, we present a simple paradigm for making occlusions effective on ImageNet for sufficiently capable models (e.g., ResNet50) that can be easily added into existing training paradigms.

References

[1] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv*, 2017. 1, 2, 3

[2] R. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. CVPR*, 2017. 1

[3] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *Proc. NeurIPS*, 2018. 1, 2, 6, 7, 8

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 1, 3

[5] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry. Augment your batch: better training with larger batches. *arXiv*, 2019. 1, 2, 4, 7, 8

[6] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html*, 2014. 1, 2

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NeurIPS*, 2012. 1, 3

[8] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *Proc. ICLR*, 2017. 2

[9] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proc. ECCV*, 2018. 8

[10] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018. 2

[11] V. Petsiuk, A. Das, and K. Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proc. BMVC*, 2018. 1

[12] S. Rebuffi, R. Fong, X. Ji, H. Bilen, and A. Vedaldi. Normgrad: Finding the pixels that matter for training. *arXiv*, 2019. 2, 3, 4, 5, 6, 7

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 1, 2

[14] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proc. ICCV*, 2017. 2, 3

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015. 3

[16] K. K. Singh and Y. J. Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Proc. ICCV*, 2017. 1, 2, 3, 5

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014. 2, 7, 8

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. 2016. 2, 7, 8

[19] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *Proc. CVPR*, 2015. 2, 7, 8

[20] X. Wang, A. Shrivastava, and A. Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proc. CVPR*, 2017. 1, 5

[21] Y. Wei, J. Feng, X. Liang, M.-M. Cheng, Y. Zhao, and S. Yan. Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1568–1576, 2017. 1, 2, 5

[22] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014. 1

[23] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proc. CVPR*, 2016. 2

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proc. CVPR*, 2018. 2

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Occlusions for Effective Data Augmentation in Image Classification
Publication Status	Published
Publication Details	Ruth C. Fong and Andrea Vedaldi. Occlusions for Effective Data Augmentation in Image Classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshop on Interpreting and Explaining Visual Artificial Intelligence Models*, 2019.

Student Confirmation

Student Name:	RUTH FONG		
Contribution to the Paper	R.C.F. proposed the initial idea, developed it into a working algorithm, ran experiments, and generated figures. R.C.F. and A.V. designed research and wrote the paper text.		
Signature 		Date	6 April 2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: ANDREA VEDALDI		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

C.2 There and Back Again: Revisiting Backpropagation Saliency Methods

The following paper was presented (virtually) at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) at Seattle, WA, USA in 2020 (Rebuffi et al., 2020).[1]

In Rebuffi et al., 2020, we make four contributions. First, introduce a novel framework for understanding propagation-based attribution methods. Second, we present a new attribution method called **NormGrad** that properly accounts for the contributions of convolutional weights. Third, we systematically study the effects of combining attribution heatmaps that were computed at different layers in a network. Lastly, we introduce a novel technique based on meta-learning to increase the class sensitivity of any propagation-based attribution method.

[1]For the appendix, see the full arXiv paper here: `https://arxiv.org/abs/2004.02866`

There and Back Again: Revisiting Backpropagation Saliency Methods

Sylvestre-Alvise Rebuffi * Ruth Fong* Xu Ji* Andrea Vedaldi
Visual Geometry Group, University of Oxford
{srebuffi, ruthfong, xuji, vedaldi}@robots.ox.ac.uk

Abstract

Saliency methods seek to explain the predictions of a model by producing an importance map across each input sample. A popular class of such methods is based on backpropagating a signal and analyzing the resulting gradient. Despite much research on such methods, relatively little work has been done to clarify the differences between such methods as well as the desiderata of these techniques. Thus, there is a need for rigorously understanding the relationships between different methods as well as their failure modes. In this work, we conduct a thorough analysis of backpropagation-based saliency methods and propose a single framework under which several such methods can be unified. As a result of our study, we make three additional contributions. First, we use our framework to propose NormGrad, a novel saliency method based on the spatial contribution of gradients of convolutional weights. Second, we combine saliency maps at different layers to test the ability of saliency methods to extract complementary information at different network levels (e.g. trading off spatial resolution and distinctiveness) and we explain why some methods fail at specific layers (e.g., Grad-CAM anywhere besides the last convolutional layer). Third, we introduce a class-sensitivity metric and a meta-learning inspired paradigm applicable to any saliency method for improving sensitivity to the output class being explained.

1. Introduction

The adoption of deep learning methods by high-risk applications, such as healthcare and automated driving, gives rise to a need for tools that help machine learning practitioners understand model behavior. Given the highly-parameterized, opaque nature of deep neural networks, developing such methods is non-trivial, and there are many possible approaches. In the basic case, even the predictions of the model itself, either unaltered or after being distilled into a simpler function [11, 3], can be used to shed light on its behaviour.

Saliency is the specific branch of interpretability concerned with determining not what the behaviour of a model is for whole input samples, but which parts of samples contribute the most to that behaviour. Thus by definition, determining saliency - or attribution - necessarily involves reversing the model's inference process in some manner [22]. Propagating a signal from the output layer of a neural network model back to the input layer is one way of explicitly achieving this.

The number of diverse works based on using signal backpropagation for interpretability in computer vision [39, 29, 4, 40] is testimony to the power of this simple principle. Typically, these techniques produce a heatmap for any given input image that ranks its pixels according to some metric of importance for the model's decision. Inspired by the work of [1], we propose to delve deeper into such methods by discussing some of the similarities, differences and potential improvements that can be illustrated.

We begin with presenting a framework that unifies several backpropagation-based saliency methods by dividing the process of generating a saliency map into two phases: *extraction* of the contribution of the gradient of network parameters at each spatial location in a particular network layer, and *aggregation* of such spatial information into a 2D heatmap. GradCAM [29], linear approximation [18] and gradient [31] can all be cast as such processes. Noting that no appropriate technique has yet been proposed for properly aggregating contributions from convolutional layers, we introduce NormGrad, which uses the Frobenius norm for aggregation. We introduce identity layers to allow for the generation of saliency heatmaps at all spatially-grounded layers in the network (i.e. even after ReLU), since NormGrad computes saliency given a parameterised network layer.

We conduct a thorough analysis of backpropagation-based saliency methods in general, with evaluation based on utilising saliency heatmaps for weak localisation. Notably, we conduct an investigation into simple techniques for combining saliency maps taken from different network layers - in contrast to the popular practice of computing maps at the input layer [31] - and find that using a weighted aver-

*indicates equal contribution

age of maps from all layers consistently improves performance for several saliency methods, compared to taking the single best layer. However, not all layers are equally important, as we discover that models optimised on datasets such as ImageNet [28] and PASCAL VOC [7] learn features that become increasingly *class insensitive* closer towards the input layer. This provides an explanation for why Grad-CAM [29] produces unmeaningful saliency heatmaps at certain layers earlier than the last convolutional layer, as the sensitivity of the gradient to class across spatial locations is eliminated by Grad-CAM's spatial gradient averaging, meaning such layers are devoid of class sensitive signals from which to form saliency heatmaps.

Finally, building off [22, 1], we introduce a novel metric for quantifying the class sensitivity of a saliency method, and present a meta-learning inspired paradigm that increases the class sensitivity of any method by adding an inner SGD step into the computation of the saliency heatmap.

2. Related work

Saliency methods. Our work focuses on backpropagation-based saliency methods; these techniques are computationally efficient as they only require one forward and backward pass through a network. One of the earliest methods was [31] which visualised the gradient at the input with respect to an output class being explained. Several authors have since proposed adaptations in order to improve the heatmap's visual quality. These include modifying the ReLU derivatives (Deconvnet [39], guided backprop [34]) and averaging over randomly perturbed inputs (SmoothGrad [33]) to produce masks with reduced noise. Several works have explored visualizing saliency at intermediate layers by combining information from activations and gradients, notably CAM [41], Grad-CAM [29], and linear approximation (a.k.a. gradient $\times$ input) [18]. Conservation-preserving methods (Excitation Backprop [40], LRP [4], and DeepLIFT [30]) modify the backward functions of network layers in order to "preserve" an attribution signal such that it sums to one at any point in the network. Reference-based methods average over attributions from multiple interpolations [35] between the input and a non-informative (e.g. black) reference input or use a single reference input with which to compare a backpropagated attribution signal [30].

Although we focus on backpropagation-based methods, another class of methods studies the effects that perturbations on the input induce on the output. [39] and [26] generate saliency maps by weighting input occlusion patterns by the induced changes in model output. [10, 9, 15] learn for saliency maps that maximally impact the model, and [5] trains a model to predict effective maps. LIME [27] learns linear weights that correspond to the effect of including or excluding (via perturbation) different image patches in an image. Perturbation-based approaches have also been used to perform object localisation [32, 37, 36].

Assessing and unifying saliency methods. A few works have studied if saliency methods have certain desired sensitivities (e.g., to specific model weights [1] or the output class being explained [22]) and if they are invariant to unmeaningful factors (e.g., constant shift in input intensity [17]). [22] showed that gradient [31], deconvnet [39], and guided backprop [34] tend to produce class insensitive heatmaps. [1] introduced sanity check metrics that measure how sensitive a saliency method is to model weights by reporting the correlation between a saliency map on a trained model vs. a partially randomized model.

Other works quantify the utility of saliency maps for weak localization [40, 10] and for impacting model predictions. [40] introduced Pointing Game, which measures the correlation between the maximal point extracted from a saliency map with pixel-level semantic labels. [10] extracts bounding boxes from saliency maps and measures their agreement with ground truth bounding boxes. [38] evaluates attribution methods using relative feature importance. [24] proposes a dataset designed for measuring visual explanation accuracy. [4, 26, 15] present variants of a perturbation-based evaluation metric that measures the impact of perturbing (or unperturbing for [15]) image patches in order of importance as given by a saliency map. However, these perturbed images are outside the training domain; [13] mitigates this by measuring the performance of classifiers re-trained on perturbed images (i.e., with 20% of pixels perturbed).

To our knowledge, the only work that has been done to unify saliency methods focuses primarily on "invasive" techniques that change backpropagation rules. The α-LRP variant [4] and Excitation Backprop [40] share the backpropagation rule, and DeepLIFT [30] is equivalent to LRP when 0 is used as the reference activation throughout a network. [20] unifies a few methods (e.g., LIME [27], LRP [4], DeepLIFT [30]) under the framework of additive feature attribution.

3. Method

Preliminaries. Consider a training set $\mathcal{D}$ of pairs $(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$ are (color) images and $\boldsymbol{y} \in \{1, \ldots, C\}$ their labels. Furthermore, let $\boldsymbol{y} = \Phi_\theta(\boldsymbol{x})$ be the output of a neural network model whose parameters θ are optimized using the cross-entropy loss ℓ to predict labels from images.

3.1. Extract & Aggregate framework

In most methods, the saliency map is obtained as a function of the network activations, computed in a forward pass,

and information propagated from the output of the network back to its input using the backpropagation algorithm. While some methods modify backpropagation in some way, here we are interested in those, such as gradient, linear approximation, and all variants of our proposed NormGrad saliency method, that do not.

In order to explain these "non-invasive" methods, we suggest that their saliency maps can be interpreted as a measure of how much the corresponding pixels contribute to changing the model parameters during a training step. We then propose a principled two-phase framework capturing this idea. In the extraction phase, a method isolates the contribution to the gradient from each spatial location. We use the fact that the gradient of spatially shared weights can be written as the sum over spatial locations of a function of the activation gradients and input features. In the aggregation phase, each spatial summand is converted into a scalar using an aggregation function, thus resulting in a saliency map.

Algorithm 1 Extract & Aggregate

1. **Extract.** Compute spatial contributions to the summation of the gradient of the weights.
   - Choose a layer whose parameters are shared spatially (layers from table 1).
   - Alternatively, insert an identity layer (section 3.1.2) at the targeted location.
2. **Aggregate.** Transform these spatial contributions into saliency maps using an aggregation function.
   - Norm: NormGrad (ours)
   - Voting/Summing: linear approximation [18]
   - Filtering: GradCAM [29], selective NormGrad (ours)
   - Max: Gradient backprogation [31]

3.1.1 Phase 1: Extract

We first choose a target layer in the network at which we plan to compute a saliency map. Assuming that the network is a simple chain[1], we can write $L = \ell \circ \Phi = h \circ k_{\boldsymbol{w}} \circ q$, where $k_{\boldsymbol{w}}$ is the target layer parameterised by $\boldsymbol{w}$, h is the composition of all layers that follow it (including loss ℓ), and q is the composition of layers that precede it. Then, $\boldsymbol{x}^{in} = q(\boldsymbol{x}) \in \mathbb{R}^{K\times H\times W}$ denotes the input to the target layer, and its output is given by $\boldsymbol{x}^{out} = k_{\boldsymbol{w}}(\boldsymbol{x}^{in}) \in \mathbb{R}^{K'\times H'\times W'}$.

[1]Other topologies are treated in the same manner, but the notation is more complex.

Layer	Spatial contribution	Size at each location
Bias	$\boldsymbol{g}_u^{out}$	vector: K'
Scaling	$\boldsymbol{g}_u^{out} \odot \boldsymbol{x}_u^{in}$	vector: K'
Conv $N \times N$	$\boldsymbol{g}_u^{out} \boldsymbol{x}_{u,N\times N}^{in}{}^\top$	matrix: $K' \times N^2K$

Table 1. Formulae and sizes of the spatial contributions to the gradient of the weights for layers with spatially shared parameters. $\odot$ denotes the elementwise product and $\boldsymbol{x}_{u,N\times N}^{in}$ is the N^2K vector obtained by unfolding the $N \times N$ patch around the target location.

Convolutional layers with general filter shapes. For convolutions with an $N \times N$ kernel size, we can re-write the convolution using the matrix form:

$$\mathbf{X}^{out} = \mathbf{W}\mathbf{X}_{N\times N}^{in} \tag{1}$$

where $\mathbf{X}^{out} \in \mathbb{R}^{K'\times HW}$ and $\mathbf{W} \in \mathbb{R}^{K'\times N^2K}$ are the output and filter tensors reshaped as matrices and $\mathbf{X}_{N\times N}^{in} \in \mathbb{R}^{N^2K\times HW}$ is a matrix whose column $\boldsymbol{x}_{u,N\times N}^{in} \in \mathbb{R}^{N^2K}$ contains the unfolded patches at location u of the input tensor to which filters are applied.[2] Then, the gradient w.r.t. the filter weights W is given by

$$\frac{dL}{d\mathbf{W}} = \sum_{u\in\Omega} \frac{d}{d\mathbf{W}} \left\langle \boldsymbol{g}_u^{out}, \mathbf{W}\boldsymbol{x}_{u,N\times N}^{in} \right\rangle = \sum_{u\in\Omega} \boldsymbol{g}_u^{out} \boldsymbol{x}_{u,N\times N}^{in}{}^\top, \tag{2}$$

where $\boldsymbol{g}_u^{out} = dh/d\boldsymbol{x}_u^{out}$ is the gradient of the "head" of the network. Thuis, for the convolutional layer case, the spatial summand is an outer product of two vectors; thus, the spatial contribution at each location to the gradient of the weights is a matrix of size $K' \times N^2K$.

Other layer types. Besides convolutional layers, bias and scaling layers also share their parameters spatially. In modern architectures, these are typically used in batch normalization layers [14]. We denote $\boldsymbol{b}, \boldsymbol{\alpha} \in \mathbb{R}^K$ as the parameters for the bias and scaling layers respectively. They are defined respectively as follows:

$$x_{ku}^{out} = x_{ku}^{in} + b_k, \qquad x_{ku}^{out} = \alpha_k x_{ku}^{in}.$$

Then, the gradients w.r.t. parameters are given by

$$\frac{dL}{d\boldsymbol{b}} = \sum_{u\in\Omega} \boldsymbol{g}_u^{out}, \qquad \frac{dL}{d\boldsymbol{\alpha}} = \sum_{u\in\Omega} \boldsymbol{g}_u^{out} \odot \boldsymbol{x}_u^{in}. \tag{3}$$

where $\odot$ is the elementwise product. For these two types of layer, the spatial summand is a vector of size K, the number of channels. Table 1 summarizes the spatial contributions to the gradients for the different layers.

[2]This operation is called `im2row` in MATLAB or `unfold` in PyTorch.

3.1.2 Virtual identity layer

So far, we have only extracted spatial gradient contributions for layers that share parameters spatially. We will now extend our summand extraction technique to any location within a CNN by allowing the insertion of a virtual identity layer at the target location. This layer is a conceptual construction that we introduce to derive in a rigorous and unified manner the equations employed by various methods to compute saliency.

We are motivated by the following question: if we were to add an identity operator at a target location, how should this operator's parameters be changed? A virtual identity layer is a layer which shares its parameters spatially and is set to the identity. Hence, it could be any of the layer from table 1, like a bias or a scaling layer; then, $b_k = 0$ or $c_k = 1$ respectively for $k \in \{1, \dots, K\}$. It could also be a $N \times N$ convolutional layer with filter bank $\mathbf{W} \in \mathbb{R}^{K \times N^2 K}$ such that $w_{kk'ij} = \delta_{k,k'}\delta_{i,0}\delta_{j,0}$, where $\delta_{a,b}$ is the Kronecker delta function[3], for $(i,j) \in \{-\frac{N-1}{2}, \dots, \frac{N-1}{2}\}$.

Because of the nature of the identity, this layer does not change the information propagated in either the forward or backward direction. Conceptually, it is attached to the part of the model that one wishes to inspect as shown in fig. 1. This layer is never "physically" added to the model (i.e., the model is not modified); its "inclusion" or "exclusion" simply denotes whether we are using input or output activations ($\boldsymbol{x}^{in}$ or $\boldsymbol{x}^{out}$). Indeed the backpropagated gradient $\boldsymbol{g}^{out}$ at the output of the identity as shown in fig. 1 is the gradient that would have been at the output of the layer preceding the identity. This construction allows to examine activations and gradients at the same location (e.g., $\boldsymbol{x}^{out}$ and $\boldsymbol{g}^{out}$) as most existing saliency methods do. We now can use the formulae defined in section 3.1.1 when analyzing the gradient of the weights of this identity layer. For example, if we consider an identity scaling layer, the spatial contribution would be $\boldsymbol{g}_u^{out} \odot \boldsymbol{x}_u^{out}$. Or, for a $N \times N$ identity convolutional layer, it would be $\boldsymbol{g}_u^{out} \boldsymbol{x}_{u,N\times N}^{out}{}^{\top}$.

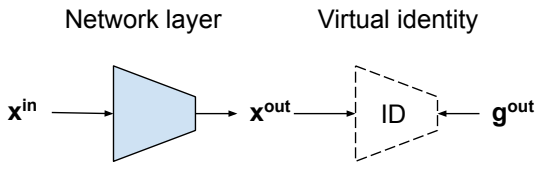


Figure 1. **Virtual identity.** Here, we visualize inserting an identity after a specific layer in the network for saliency computation purposes. The gradient coming from later stages of the network that is the gradient at output of the network layer, is now also the gradient at the output of the identity, $\boldsymbol{g}^{out}$.

[3]Kronecker delta function: $\delta_{a,b} = 1$ if $a = b$; otherwise, $\delta_{a,b} = 0$.

3.1.3 Phase 2: Aggregate

Following the extraction phase (section 3.1.1), we have the local contribution at each spatial location to the gradients of either an existing layer or a virtual identity layer. Each spatial location is associated with a vector or matrix (table 1). In this section, we describe different aggregation functions to map these vectors or matrices to a single scalar per spatial position. We drop the spatial location u in the notations.

Understanding existing saliency methods. One possible aggregation function is the sum of the elements in a vector. When combined with a virtual scaling identity layer (section 3.1.2), we obtain the linear approximation method [18]: $\sum_k g_k^{out} x_k^{out}$.The contributions from the scaling identity encode the result of channels changing (after the gradient is applied) at each location; thus, the sum aggregation function acts as a voting mechanism. The resulting saliency map highlights the locations that would be most impacted if following through with the channel updates.

Aggregation functions can also be combined with element-wise filtering functions (e.g., the absolute value function). Another aggregation function takes the maximum absolute value of the vector: $\mathrm{maxabs}(\boldsymbol{x}) = \max_k |x_k|$. If we combine a virtual bias identity layer in phase 1, which gives $\boldsymbol{g}^{out}$ as the spatial contribution, with the maxabs function for aggregation, we obtain at each spatial location the gradient [31] method: $\max_k |g_k^{out}|$.

As for CAM [41] and Grad-CAM [29], we cannot directly use the spatial contributions extracted at each location because they spatially average $\boldsymbol{g}^{out}$.[4] However, for architectures that use global average pooling followed after their convolutional layers (e.g., ResNet architectures), $\boldsymbol{g}_u^{out} = \bar{\boldsymbol{g}}^{out}$. Then, CAM and Grad-CAM and CAM can be viewed as combining a virtual scaling identity layer from phase 1 with summing and positive filtering (i.e., $\mathrm{filter}_+(x) = (x)_+$) functions for aggregation.

NormGrad. The sum and max functions have clear interpretations when using bias or scaling identity layers; however, they cannot be easily transported to convolutional identity layers as interactions between input channels can vary depending on the output channel and are represented as a matrix, not as a vector as are the case for scaling and bias layers. Thus, we would like to have an aggregation function that could be used to aggregate any type of spatial contribution, regardless of its shape.

Using the norm function satisfies this criterion, for example the L^2 norm when dealing with vectors and the Frobenius norm for matrices. We note that the matrices obtained at each location for convolutional layers are the outer

[4]Because CAM global average pooling + one fully connected layer, $\boldsymbol{g}^{out}$ is equal to the fully connected weights.

Phase 1	Phase 2	Saliency map
Bias IDL	Max	$\max_k g_k^{out}$
Scaling IDL	Sum	$\sum_k g_k^{out} x_k^{out}$
Scaling IDL	F + N	$\|(g^{out} \odot x^{out})_+\|$
Conv 1×1 IDL	Norm	$\|\boldsymbol{g}^{out}\|\|\boldsymbol{x}^{out}\|$
Real Conv 3×3	Norm	$\|\boldsymbol{g}^{out}\|\|\boldsymbol{x}^{in}_{3\times3}\|$

Table 2. **Combining layers and aggregation functions for saliency.** g_k^{out} and x_k^{out} are tensor slices for channel k and contain only spatial information. IDL denotes an identity layer. F + N is positive filtering + norm. From top to bottom, the rows correspond to the following saliency methods: 1., gradient, 2., linear approximation, 3., NormGrad selective, 4., NormGrad, and 5., NormGrad without the virtual identity trick.

products of two vectors. For such matrices, the Frobenius norm is equal to the product of the norms of the two vectors. For example, for an existing 1×1 convolutional layer, we consider the saliency map given by $\|\boldsymbol{g}^{out}\|\|\boldsymbol{x}^{in}\|$. We call this class of saliency methods derived by using norm as an aggregation function, "NormGrad".

Saliency maps from the NormGrad outlined above are not as class selective as other methods because they highlight the spatial locations that contribute the most to gradient of the weights, both positively and negatively. One way to introduce class selectivity is to use positive filtering before applying the norm. If we apply norm and positive filtering aggregation to a scaling identity layer, the resulting saliency map is given by $\|(\boldsymbol{g}^{out} \odot \boldsymbol{x}^{out})_+\|$. Throughout the rest of the paper, we call this variant "selective NormGrad".

4. Experiments

In this section we quantitatively and qualitatively evaluate the performance of a large number of backpropagation-based saliency methods. Code for our framework is released at http://github.com/srebuffi/revisiting_saliency. Additional experiments such as image captioning visualizations are included in the appendix.

Experimental set-up. Unless otherwise stated, saliency maps are generated on images from either the PASCAL VOC [7] 2007 test set or the ImageNet [6] 2012 validation set for either VGG16 [31] or Resnet50 [12]. For PASCAL VOC, we use a model pre-trained on ImageNet with fine-tuned fully connected layers on PASCAL VOC. We use [9]'s TorchRay interpretability package to generate saliency methods for all other saliency methods besides our NormGrad and meta-saliency methods as well as to evaluate saliency maps on [40]'s Pointing Game (see [40] for more details). For all correlation analysis, we compute the Spearman's correlation coefficient [23] between saliency maps that are upsampled to the input resolution: 224×224.

4.1. Justifying the virtual identity trick.

In order to justify our use of the virtual identity trick, we compare NormGrad saliency maps generated at 1×1 and 3×3 convolutional layers both with and without the virtual identity trick (4th and 5th rows respectively in table 2) for VGG16 and ResNet50. We first computed the correlations between saliency maps generating with and without the virtual identity trick. We found that the mean correlation across $N = 50k$ ImageNet validation images was over 95% across all layers we tested. We also evaluated their performance on the Pointing Game [40] and found that the mean absolute difference in pointing game accuracy was $0.53\% \pm 0.62\%$ across all layers we tested (see supp. for more details and full results). This empirically demonstrates that using the virtual identity trick closely approximates the behavior of calculating the spatial contributions for the original convolutional layers.

4.2. Combining saliency maps across layers.

Training linear classifiers on top of intermediate representations is a well-known method for evaluating the learned features of a network at different layers [2]. This suggests that saliency maps, too, may have varying levels of meaningfulness at different layers.

We explore this question by imposing several weighting methods for combining the layer-wise saliency maps of ResNet50 and VGG16 and measuring the resulting performance on PASCAL VOC Pointing Game on both the "difficult" and "all" image sets. To determine the weight γ_j for a given layer j out of J layers in a network we use:

1. **Feature spread**. Given the set of feature activations at layer j, $\boldsymbol{x}^i$ for $i \in M$ input images sampled uniformly across classes, compute the spatial mean $\bar{\boldsymbol{x}}^i = \sum_{u \in \Omega} \boldsymbol{x}^i_u$. $\gamma_j = \frac{1}{M} \sum_{i=1}^{M} |\bar{\boldsymbol{x}}^i - \bar{\boldsymbol{x}}^\mu|$ where $\bar{\boldsymbol{x}}^\mu = \frac{1}{M} \sum_{i=1}^{M} |\bar{\boldsymbol{x}}^i|$.
2. **Classification accuracy**. Given the set of feature activations at layer j, $\boldsymbol{x}^i$ for $i \in M$ input images sampled uniformly across classes, train a linear layer Ψ using image labels $\boldsymbol{y}^i$. $\gamma_j = \frac{1}{M} \sum_{i=1}^{M} \delta_{\Psi(\boldsymbol{x}^i), \boldsymbol{y}^i}$ where $\delta_{\Psi(\boldsymbol{x}^i), \boldsymbol{y}^i} = 1$ if $\Psi(\boldsymbol{x}^i) = \boldsymbol{y}^i$, 0 otherwise.
3. **Linear interpolation**. $\gamma_j = \frac{j}{J}$.
4. **Uniform**. $\gamma_j = \frac{1}{J}$.

To obtain a combined saliency map $\boldsymbol{M}$ from maps $\boldsymbol{m}_j$ from each layer j, the weights are normalised and applied either additively, $\boldsymbol{M} = \sum_{j=0}^{J} \gamma_j \cdot \boldsymbol{m}_j$, or with a product, $\boldsymbol{M} = \prod_{j=0}^{J} \boldsymbol{m}_j^{\gamma_j}$ (see fig. 2 for visualization).

Table 3 shows that weighted saliency maps produce the best overall performance in all four key test cases, which is surprising as there were only four weight schemes tested

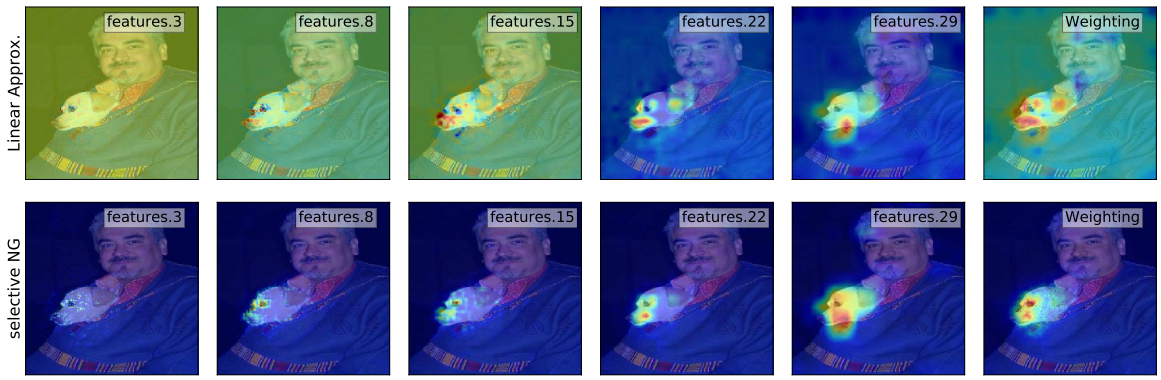


Figure 2. **Saliency maps at different network depths and as a weighted combination.** Linear approximation vs. selective NormGrad saliency maps of VGG16 on VOC2007. The first 5 images of each row correspond to different depths whereas the last one is a weighted product combination (using classification accuracy weights) of the first saliency maps. We observe that the weighted version produces more fined grained maps for both methods.

	All				Difficult			
	Resnet50		VGG16		Resnet50		VGG16	
	b.s.	b.w.	b.s.	b.w.	b.s.	b.w.	b.s.	b.w.
CEB	**90.7**	88.6	82.1	78.2	82.2	82.2	67.0	65.2
EB	84.5	83.1	77.5	75.7	71.5	71.3	57.8	56.1
GC	90.3	90.5	**86.6**	80.6	**82.3**	82.6	74.0	67.8
Gd	83.9	83.3	**86.6**	82.7	70.3	69.4	66.4	67.4
Gds	80.0	77.4	76.8	77.2	62.9	59.5	57.9	59.4
Gui	82.3	81.0	75.8	74.4	67.9	63.4	53.0	51.6
LA	90.2	**91.2**	86.4	**86.9**	81.9	**83.8**	**74.5**	**77.4**
NG	84.6	83.5	81.9	81.8	72.2	70.2	64.8	64.6
sNG	87.4	88.7	86.0	86.8	77.0	79.1	72.6	74.5

Table 3. **Pointing game results on VOC07.** b.s. and b.w. stand for best single layer and best weighted combination. (C)EB: (Contrastive) Excitation Backprop, GC: GradCAM, Gd(s): Gradient (sum), Gui: guided backprop, LA: linear approximation, (s)NG: (selective) NormGrad.

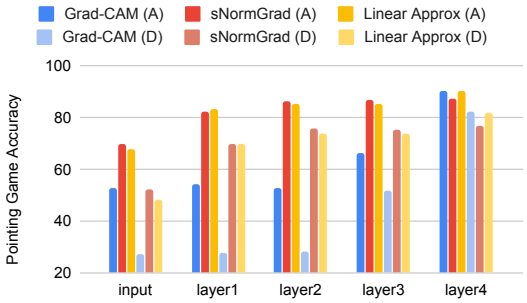


Figure 3. **Select Pointing Game results.** Results for ResNet50 on VOC07 at different network depths (A: all images; D: difficult subset). Grad-CAM performs worse at every layer except the last conv layer and lower than pointing at the center (all: 69.6%; diff: 42.4%) at most layers.

(in addition to best single layer), none of which were explicitly optimised for use with saliency maps. Our results strongly indicate that linear approximation in particular benefits from combining maps from different layers, and linear approximation with layer combination consistently produces the best performance overall and beats far more complex methods at weak localisation using a single forward-backward pass (see supp. for full results).

Note that the feature spread and classification accuracy metrics can both be used as indicators of class sensitivity (section 4.3). This is because if feature activations are uniform for images sampled across classes, it is not possible for them to be sensitive to - or predictive of - class, and the classification accuracy metric is an explicit quantisation of how easily features can be separated into classes. We observe from the computed weights that both metrics generally increase with layer depth (see supp.).

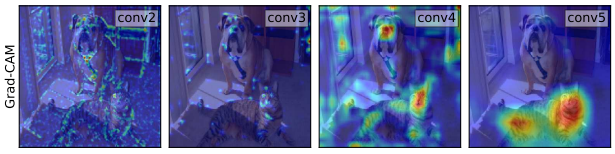


Figure 4. **Grad-CAM failure mode.** Grad-CAM saliency maps w.r.t. "tiger cat" at different depths of VGG16. Grad-CAM only works at the last conv layer (rightmost col).

Explanation of Grad-CAM failure mode. Figure 4 showed qualitatively that Grad-CAM does not produce meaningful saliency maps at any layer except the last convolutional layer, which is confirmed by Grad-CAM's Pointing Game results at earlier layers. Class sensitivity - as measured by our weighting metrics - increasing with layer depth offers an explanation for this drop in performance. Since

Grad-CAM spatially averages the backpropagated gradient before taking a product with activations, each pixel location in the heatmap receives the same gradient vector (across channels) *irrespective* of the image content contained within its receptive field. Thus, if the activation map used in the ensuing product is also not class selective - firing on both dogs and cats for example, fig. 4 - the saliency map cannot be. On the other hand, methods that do not spatially average gradients such as NormGrad (fig. 3) can rely on gradients that are free to vary across the heatmap with underlying class, increasing the class sensitivity of the resulting saliency map.

4.3. An explicit metric for class sensitivity

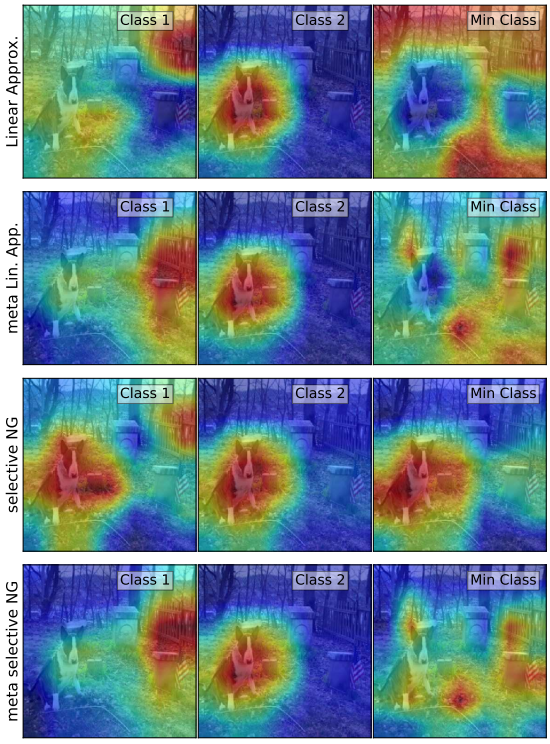


Figure 5. **Class sensitivity with and without meta-saliency.** Min class saliency maps that use meta-saliency (row 2 and 4, right col) are less informative than those that don't use meta-saliency (rows 1 and 3, right col). Class 1 is the ground truth class (fence), class 2 is the maximally predicted class (Cardigan Welsh corgi), min class is the minimally predicted class (black widow spider).

[22] qualitatively shows that early backprop-based methods (e.g., gradient, deconvnet, and guided backprop) are not sensitive to the output class being explained by showing that saliency maps generated w.r.t. different output classes and gradient signals appear visually indistinguishable. Thus, similar to [1], we introduce a sanity check to measure a saliency method's output class sensitivity. We compute the correlation between saliency maps w.r.t. to output class predicted with highest confidence (max class) and that predicted with lowest confidence (max and min class respectively) for $N = 1000$ ImageNet val. images (1 per class).

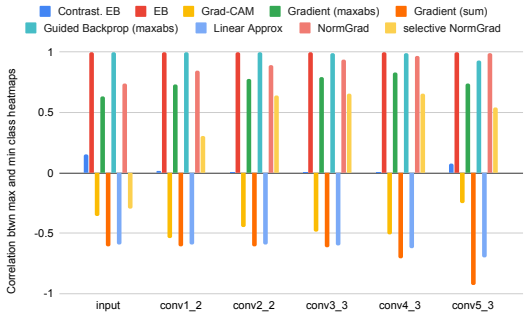


Figure 6. **Class sensitivity of saliency methods.** This plot shows the correlation between VGG16 saliency maps computed w.r.t. to the maximally and minimally predicted class (closer to zero is better).

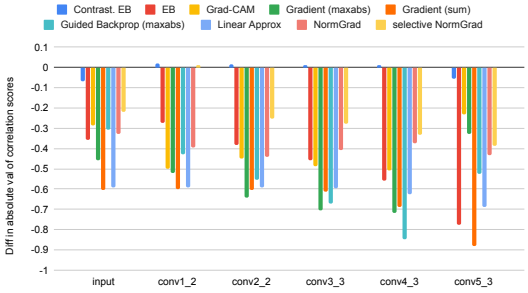


Figure 7. **Meta-saliency improves class sensitivity for all saliency methods.** Using meta-saliency yields weaker correlations between the saliency maps w.r.t. the maximally and minimally predicted output class compared to not using meta-saliency (lower is better).

We would expect saliency maps w.r.t. the max class to be visually salient while those w.r.t. to the min class to be uninformative (because the min class is not in the image). Thus, we desire the correlation scores to be close to zero.

Figure 6 shows results for various saliency methods. We observe that excitation backprop and guided backprop yield correlation scores close to 1 for all layers, while contrastive excitation backprop yields scores closest to 0. Furthermore, methods using sum aggregation (e.g., gradient [sum], linear approx, and Grad-CAM) have negative scores (i.e., their max-min-class saliency maps are anti-correlated). This is because sum aggregation acts as a voting mechanism; thus, these methods reflect the fact that the network has learned anti-correlated relationships between max and min classes.

4.4. Meta-saliency analysis

As a general method for improving the sensitivity of saliency heatmaps to the output class used to generate the

gradient, we propose to perform an inner SGD step before computing the gradients with respect to the loss. This way we can extend any saliency method to second order gradients. This is partly inspired by the inner step used in, for example, few shot learning [8] and architecture search [19]. We want to minimize:

$$L(\theta, x) = \ell(\theta - \epsilon \nabla_\theta \ell(\theta, x), x). \tag{4}$$

We take $\epsilon \ll 1$ to use a Taylor expansion of this loss at θ and we now have the resulting approximated loss:

$$L(\theta, x) \approx \ell(\theta, x) - \epsilon \|\nabla_\theta \ell(\theta, x)\|^2. \tag{5}$$

As done in the previous section, we can now take the gradient of the loss with respect to the parameters θ:

$$\nabla_\theta L(\theta, x) \approx \nabla_\theta \ell(\theta, x) - 2\epsilon \nabla^2_\theta \ell(\theta, x) \nabla_\theta \ell(\theta, x). \tag{6}$$

Using a finite difference scheme of step h as in [25], we can approximate the hessian-vector product by:

$$\nabla^2_\theta \ell(\theta, x) \nabla_\theta \ell(\theta, x) = \frac{\nabla_\theta \ell(\theta, x) - \nabla_{\theta^-} \ell(\theta^-, x)}{h} + O(h).$$

where $\theta^- = \theta - h \nabla_\theta \ell(\theta, x)$. We chose on purpose a backward finite difference such that two terms cancel each other when taking $h = 2\epsilon$ and we get:

$$\nabla_\theta L(\theta, x) \approx \nabla_{\theta'} \ell(\theta', x).$$

where $\theta' = \theta - 2\epsilon \nabla_\theta \ell(\theta, x)$ corresponds to one step of SGD of learning rate 2ϵ. We notice that if we take $\epsilon \to 0$, this formula boils back down to the original gradient of the weights without meta step. We further note that this meta saliency approach only requires one more forward-backward pass compared to usual saliency backpropagation methods.

Conversely, if we would like to get an importance map that highlights the degradation of the model's performance, we should add an inner step with gradient ascent within the loss. Hence by minimizing the resulting loss $-\ell(\theta + \epsilon \nabla_\theta \ell(\theta, x), x)$, we get the same formula for the gradients of the weights but with $\theta' = \theta + 2\epsilon \nabla_\theta \ell(\theta, x)$.

We hypothesize that applying meta-saliency to a saliency method should decrease correlation strength because allowing the network to update one SGD step in the direction of the min class should "destroy" the informativeness of the resulting saliency map. We use a learning rate $\epsilon = 0.001$ for the class sensitivity quantitative analysis. Figure 5 shows qualitatively that this appears to be the case: without meta-saliency, selective NormGrad and linear approximation yield max (class 2) and min class heatmaps that are highly positively and negatively correlated respectively. However, when meta-saliency is applied, the min class saliency map appears more random. Figure 7 shows results comparing the max-min class correlation scores with and without meta-saliency. These results demonstrate that meta-saliency decreases max-min class correlation strength for nearly all saliency methods and suggest that meta-saliency can increase the class sensitivity for any saliency method.

4.5. Model weights sensitivity

[1] shows that some saliency methods (e.g., Guided Backprop in particular) are not sensitive to model weights as they are randomized in a cascading fashion from the end to the beginning of the network. Figure 8 shows qualitatively that, by the late conv layers, saliency maps for linear approximation and selective NormGrad are effectively scrambled (top two rows). It also highlights that, because meta-saliency increases class selectivity and is allowed to take one SGD in the direction of the target class, it takes relatively longer (i.e., more network depth) to randomize a meta-saliency heatmap (bottom row and see appendix).

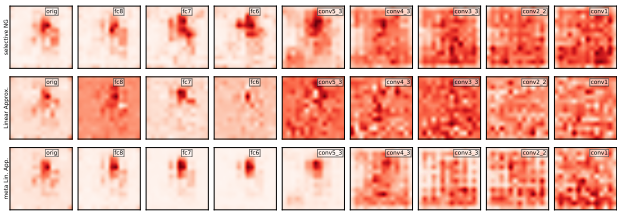


Figure 8. **Model weights sensitivity.** Sanity check by randomizing VGG16 model weights in a cascading fashion for the "Irish terrier" image from [1]. Top row: selective NormGrad, middle row: linear approximation, bottom row: linear approximation with meta-saliency (lower correlation with orig heatmap [leftmost col] is better). All methods look random after conv4_3. By comparing the last two rows at conv5_3, we see the that meta-saliency enforces more class sensitivity than the non-meta variant.

5. Conclusions

We introduced a principled framework based on the contribution of each spatial location to the weights' gradient. This framework unifies several existing backpropagation-based methods and allowed us to systematically explore the space of possible saliency methods. We use it for example to formulate NormGrad, a novel saliency method. We also studied how to combine saliency maps from different layers, discovering that it can consistently improve weak localization performance and produce high resolution maps. Finally, we introduced a class-sensitivity metric and proposed meta-saliency, a novel paradigm applicable to any existing method to improve sensitivity to the target class.

6. Acknowledgments

This work is supported by Mathworks/DTA, Open Philanthropy, EPSRC AIMS CDT and ERC 638009-IDIU.

References

[1] Julius Adebayo, Justin Gilmer, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proc. NeurIPS*, 2018. 1, 2, 7, 8, 12

[2] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv*, 2016. 5

[3] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proc. NIPS*, 2014. 1

[4] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 2015. 1, 2

[5] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Proc. NIPS*, 2017. 2

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009. 5

[7] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 2015. 2, 5

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, 2017. 8

[9] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *Proc. ICCV*, 2019. 2, 5, 11

[10] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. CVPR*, 2017. 2

[11] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv*, 2017. 1

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 5

[13] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. Evaluating feature importance estimates. *arXiv*, 2018. 2

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015. 3

[15] Andrei Kapishnikov, Tolga Bolukbasi, Fernanda Viégas, and Michael Terry. Xrai: Better attributions through regions. In *Proc. ICCV*, 2019. 2

[16] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proc. CVPR*, 2015. 12

[17] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. 2019. 2

[18] Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv*, 2016. 1, 2, 3, 4

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv*, 2018. 8

[20] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proc. NIPS*, 2017. 2

[21] Ruotian Luo, Brian Price, Scott Cohen, and Gregory Shakhnarovich. Discriminability objective for training descriptive captions. *arXiv*, 2018. 12

[22] A. Mahendran and A. Vedaldi. Salient deconvolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 1, 2, 7

[23] Leann Myers and Maria J Sirois. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences*, 12, 2004. 5

[24] Jose Oramas, Kaili Wang, and Tinne Tuytelaars. Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks. In *Proc. ICLR*, 2019. 2

[25] Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 1994. 8

[26] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proc. BMVC*, 2018. 2

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proc. KDD*, 2016. 2

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 2

[29] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proc. ICCV*, 2017. 1, 2, 3, 4, 12

[30] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proc. ICML*, 2017. 2

[31] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. ICLR*, 2014. 1, 2, 3, 4, 5

[32] Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Proc. ICCV*, 2017. 2

[33] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv*, 2017. 2

[34] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv*, 2014. 2

[35] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proc. ICML*, 2017. 2

[36] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proc. CVPR*, 2017. 2

[37] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining

with adversarial erasing: A simple classification to semantic segmentation approach. In *Proc. CVPR*, 2017. 2

[38] Mengjiao Yang and Been Kim. Benchmarking attribution methods with relative feature importance. *arXiv*, 2019. 2

[39] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014. 1, 2

[40] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *Proc. ECCV*, 2016. 1, 2, 5, 11

[41] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proc. CVPR*, 2016. 2, 4

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	There and Back Again: Revising Backpropagation Saliency Methods
Publication Status	Accepted for Publication
Publication Details	Sylvestre-Alvise Rebuffi*, Ruth Fong*, Xu Ji*, and Andrea Vedaldi. There and Back Again: Revisiting Backpropagation Saliency Methods. To appear in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. * denotes equal contribution.

Student Confirmation

Student Name:	RUTH FONG		
Contribution to the Paper	S.R., R.F., and X.J. jointly designed research, ran experiments, and generated figures. S.R. came up with the initial idea for the Extract & Aggregate framework and the NormGrad formulation. R.F. came up with the initial idea for the class sensitivity analysis and metric. X.J. came up with the idea for combining saliency maps at different layers. S.R., R.F., X.J., and A.V. jointly wrote the paper text.		
Signature		Date	6 April 2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: ANDREA VEDALDI		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

Index

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng (2016). "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 265–283.

J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim (2018). "Sanity checks for saliency maps". In: *Proc. NeurIPS*.

P. Agrawal, R. Girshick, and J. Malik (2014). "Analyzing the performance of multilayer neural networks for object recognition". In: *Proc. ECCV*.

G. Alain and Y. Bengio (2016). "Understanding intermediate layers using linear classifier probes". In: *arXiv preprint arXiv:1610.01644*.

A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al. (2019). "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI". In: *arXiv preprint arXiv:1910.10045*.

S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek (2015). "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PloS one* 10.7, e0130140.

D. Bau, H. Strobelt, W. Peebles, J. Wulff, B. Zhou, J. Zhu, and A. Torralba (2019a). "Semantic Photo Manipulation with a Generative Image Prior". In: *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* 38.4.

D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba (2017). "Network Dissection: Quantifying Interpretability of Deep Visual Representations". In: *Proc. CVPR*.

D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman, and A. Torralba (2019b). "GAN Dissection: Visualizing and Understanding Generative Adversarial Networks". In: *Proc. ICLR*.

P. E. Black (2005). *Dictionary of Algorithms and Data Structures, US National Institute of Standards and Technology*.

J. K. Blitzstein and J. Hwang (2019). *Introduction to Probability*. Crc Press. URL: `http://probabilitybook.net`.

T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). "Man is to computer programmer as woman is to homemaker? Debiasing word embeddings". In: *Proc. NeurIPS*.

W. Brendel and M. Bethge (2019). "Approximating CNNs with bag-of-local-features models works surprisingly well on Imagenet". In: *Proc. ICLR*.

J. Brownlee (2018). *Basics of Mathematical Notation for Machine Learning*. Online. URL: `https://machinelearningmastery.com/basics-mathematical-notation-machine-learning`.

M. Brundage, S. Avin, J. Wang, H. Belfield, G. Krueger, G. Hadfield, H. Khlaaf, J. Yang, H. Toner, R. Fong, et al. (2020). "Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims". In: *arXiv preprint arXiv:2004.07213*.

J. Buolamwini and T. Gebru (2018). "Gender shades: Intersectional accuracy disparities in commercial gender classification". In: *Conference on Fairness, Accountability and Transparency*.

C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, et al. (2015). "Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks". In: *Proc. ICCV*.

Captum (2019). Online. URL: `https://github.com/pytorch/captum`.

S. Carter, Z. Armstrong, L. Schubert, I. Johnson, and C. Olah (2019). "Activation atlas". In: *Distill* 4.3, e15.

S. Carter, D. Ha, I. Johnson, and C. Olah (2016). "Experiments in handwriting with a neural network". In: *Distill* 1.12, e4.

C.-H. Chang, E. Creager, A. Goldenberg, and D. Duvenaud (2017). "Interpreting neural network classifications with variational dropout saliency maps". In: *Proc. NeurIPS*.

C.-H. Chang, E. Creager, A. Goldenberg, and D. Duvenaud (2019). "Explaining image classifiers by counterfactual generation". In: *Proc. ICLR*.

C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su (2019). "This looks like that: deep learning for interpretable image recognition". In: *Proc. NeurIPS*.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel (2016). "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Proc. NeurIPS*.

X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille (2014). "Detect what you can: Detecting and representing objects using holistic models and body parts". In: *Proc. CVPR*.

Y. Cheng, D. Wang, P. Zhou, and T. Zhang (2017). "A survey of model compression and acceleration for deep neural networks". In: *arXiv preprint arXiv:1710.09282*.

R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva (2016). "Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence". In: *Scientific reports* 6, p. 27755.

P. Dabkowski and Y. Gal (2017). "Real time image saliency for black box classifiers". In: *Proc. NeurIPS*.

K. E. David, Q. Liu, and R. Fong (2020). "Debiasing Convolutional Neural Networks via Meta Orthogonalization". In: *Proc. NeurIPS workshop on Algorithmic Fairness through the Lens of Causality and Interpretability*.

T. DeVries and G. W. Taylor (2017). "Improved regularization of convolutional neural networks with cutout". In: *arXiv preprint arXiv:1708.04552*.

A. Dosovitskiy and T. Brox (2016a). "Generating images with perceptual similarity metrics based on deep networks". In: *Proc. NeurIPS*.

A. Dosovitskiy and T. Brox (2016b). "Inverting visual representations with convolutional networks". In: *Proc. CVPR*.

M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman (Jan. 2015). "The Pascal Visual Object Classes Challenge: A Retrospective". In: *IJCV* 111.1, pp. 98–136.

R. Fong, A. Mordvintsev, A. Vedaldi, and C. Olah (in prep). *Interactive Similarity Overlays*. In preparation.

R. Fong, M. Patrick, and A. Vedaldi (2019a). "Understanding Deep Networks via Extremal Perturbations and Smooth Masks". In: *Proc. ICCV*.

R. Fong, W. Scheirer, and D. Cox (2018a). "Using human brain activity to guide machine learning". In: *Scientific Reports* 8.1, p. 5397.

R. Fong and A. Vedaldi (2017). "Interpretable Explanations of Black Boxes by Meaningful Perturbation". In: *Proc. ICCV*.

R. Fong and A. Vedaldi (2018b). "Net2Vec: Quantifying and Explaining How Concepts Are Encoded by Filters in Deep Neural Networks". In: *Proc. CVPR*.

R. Fong and A. Vedaldi (2019b). "Explanations for Attributing Deep Neural Network Predictions". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller. Springer Cham. Chap. 8.

R. Fong and A. Vedaldi (2019c). "Occlusions for Effective Data Augmentation in Image Classification". In: *Proc. ICCV workshop on Interpreting and Explaining Visua Artificial Intelligence Models*.

F. B. Fuchs, O. Groth, A. R. Kosiorek, A. Bewley, M. Wulfmeier, A. Vedaldi, and I. Posner (2019). "Scrutinizing and De-Biasing Intuitive Physics with Neural Stethoscopes". In: *Proc. BMVC*.

G. Ghiasi, T.-Y. Lin, and Q. V. Le (2018). "DropBlock: A regularization method for convolutional networks". In: *Proc. NeurIPS*.

S. Gidaris, P. Singh, and N. Komodakis (2018). "Unsupervised representation learning by predicting image rotations". In: *arXiv preprint arXiv:1803.07728*.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal (2018). "Explaining explanations: An overview of interpretability of machine learning". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, pp. 80–89.

A. Gonzalez-Garcia, D. Modolo, and V. Ferrari (2016). "Do semantic parts emerge in Convolutional Neural Networks?" In: *IJCV*.

I. J. Goodfellow, J. Shlens, and C. Szegedy (2015). "Explaining and harnessing adversarial examples". In: *Proc. ICLR*.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Proc. NeurIPS*.

S. Greydanus, A. Koul, J. Dodge, and A. Fern (2018). "Visualizing and understanding atari agents". In: *Proc. ICML*.

C. G. Gross (2002). "Genealogy of the 'grandmother cell'". In: *The Neuroscientist* 8.5, pp. 512–518.

D. Gunning (Nov. 2017). *Explainable Artificial Intelligence (XAI)*. Online. URL: `https://www.darpa.mil/attachments/XAIProgramUpdate.pdf`.

W. Hämäläinen and M. Vinni (2006). "Comparison of machine learning methods for intelligent tutoring systems". In: *International Conference on Intelligent Tutoring Systems*. Springer, pp. 525–534.

A. W. Harley (2015). "An Interactive Node-Link Visualization of Convolutional Neural Networks". In: *Proc. of International Symposium on Visual Computing (ISVC)*.

K. He, X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proc. CVPR*.

L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell (2016). "Generating visual explanations". In: *Proc. ECCV*.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner (2017). "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *Proc. ICLR*.

G. Hinton, O. Vinyals, and J. Dean (2015). "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531*.

F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker (2019). "Gamut: A Design Probe to Understand How Data Scientists Understand Machine Learning Models". In: *Proc. CHI*.

F. Hohman, N. Hodas, and D. H. Chau (2017). "ShapeShop: Towards Understanding Deep Learning Representations via Interactive Experimentation". In: *Proc. CHI Extended Abstracts*.

F. Hohman, M. Kahng, R. Pienta, and D. H. Chau (2018). "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers". In: *TCVG* 25.8, pp. 2674–2693.

D. Huk Park, L. Anne Hendricks, Z. Akata, A. Rohrbach, B. Schiele, T. Darrell, and M. Rohrbach (2018). "Multimodal explanations: Justifying decisions and pointing to the evidence". In: *Proc. CVPR*.

S. Ioffe and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167*.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093*.

M. Johnson (Dec. 2018). *Providing Gender-Specific Translations in Google Translate*. Online. URL: `https://ai.googleblog.com/2018/12/providing-gender-specific-translations.html`.

M. Johnson (Apr. 2020). *A Scalable Approach to Reducing Gender Bias in Google Translate*. Online. URL: `https://ai.googleblog.com/2020/04/a-scalable-approach-to-reducing-gender.html`.

M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg (2018). "Gan lab: Understanding complex deep generative models using interactive visual experimentation". In: *TCVG* 25.1, pp. 1–11.

D. Kang, D. Raghavan, P. Bailis, and M. Zaharia (2020). "Model Assertions for Monitoring and Improving ML Model". In: *arXiv preprint arXiv:2003.01668*.

A. Kapishnikov, T. Bolukbasi, F. Viégas, and M. Terry (2019). "XRAI: Better Attributions Through Regions". In: *Proc. ICCV*.

R. Khanna, B. Kim, J. Ghosh, and O. Koyejo (2018). "Interpreting black box predictions using fisher kernels". In: *arXiv preprint arXiv:1810.10118*.

B. Kim (2015). "Interactive and interpretable machine learning models for human machine collaboration". PhD thesis. Massachusetts Institute of Technology.

B. Kim, R. Khanna, and O. O. Koyejo (2016). "Examples are not enough, learn to criticize! criticism for interpretability". In: *Advances in neural information processing systems*, pp. 2280–2288.

B. Kim, E. Reif, M. Wattenberg, and S. Bengio (2019). "Do Neural Networks Show Gestalt Phenomena? An Exploration of the Law of Closure". In: *arXiv prepreint arXiv:1903.01069*.

B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. (2018). "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)". In: *Proc. ICML*.

P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim (2019). "The (un)reliability of saliency methods". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, pp. 267–280.

P.-J. Kindermans, K. Schütt, K.-R. Müller, and S. Dähne (2016). "Investigating the influence of noise and distractors on the interpretation of neural networks". In: *arXiv preprint arXiv:1611.07270*.

J. Konorski (1967). "Integrative activity of the brain; an interdisciplinary approach". In:

N. Kriegeskorte, M. Mur, and P. A. Bandettini (2008). "Representational similarity analysis-connecting the branches of systems neuroscience". In: *Frontiers in systems neuroscience* 2, p. 4.

A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Proc. NeurIPS*.

I. Lage, A. Ross, S. J. Gershman, B. Kim, and F. Doshi-Velez (2018). "Human-in-the-loop interpretability prior". In: *Proc. NeurIPS*.

I. Laina, R. Fong, and A. Vedaldi (2020). "Quantifying Learnability and Describability of Visual Concepts Emerging in Representation Learning". In: *Proc. NeurIPS*.

H. Lakkaraju, S. H. Bach, and J. Leskovec (2016). "Interpretable decision sets: A joint framework for description and prediction". In: *Proc. KDD*.

S. Lapuschkin, A. Binder, G. Montavon, K.-R. Muller, and W. Samek (2016). "Analyzing classifiers: Fisher vectors and deep neural networks". In: *Proc. CVPR*.

S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller (2019). "Unmasking Clever Hans predictors and assessing what machines really learn". In: *Nature communications* 10.1, pp. 1–8.

G. Larsson, M. Maire, and G. Shakhnarovich (2017). "Fractalnet: Ultra-deep neural networks without residuals". In: *Proc. ICLR*.

Y. LeCun, C. Cortes, and C. Burges (2010). *MNIST handwritten digit database*. Online. URL: `http://yann.lecun.com/exdb/mnist`.

J. Lee (2017). "A survey of robot learning from demonstrations for human-robot collaboration". In: *arXiv preprint arXiv:1710.08789*.

K. Lenc and A. Vedaldi (2015). "Understanding image representations by measuring their equivariance and equivalence". In: *Proc. CVPR*.

Loss Functions (2017). Online. URL: `https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html`.

L. v. d. Maaten and G. Hinton (2008). "Visualizing data using t-SNE". In: *JMLR*.

A. Madsen (2019). "Visualizing memorization in RNNs". In: *Distill* 4.3, e16. URL: `https://distill.pub/2019/memorization-in-rnns`.

A. Mahendran and A. Vedaldi (2015). "Understanding deep image representations by inverting them". In: *Proc. CVPR*.

A. Mahendran and A. Vedaldi (2016a). "Salient Deconvolutional Networks". In: *Proc. ECCV*.

A. Mahendran and A. Vedaldi (2016b). "Visualizing Deep Convolutional Neural Networks Using Natural Pre-images". In: *IJCV* 120.3, pp. 233–255.

D. Marcos, R. Fong, S. Lobry, R. Flamary, N. Courty, and D. Tuia (2020). "Contextual Semantic Interpretability". In: *Proc. ACCV*.

D. Marcos, S. Lobry, and D. Tuia (2019). "Semantically Interpretable Activation Maps: what-where-how explanations within CNNs". In: *Proc. ICCV workshop on Interpreting and Explaining Visua Artificial Intelligence Models*.

A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick (2018). "On the importance of single directions for generalization". In: *arXiv preprint arXiv:1803.06959*.

A. Mordvintsev (2016). *DeepDreaming with TensorFlow*. Online. URL: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb.

A. Mordvintsev, C. Olah, and M. Tyka (2015). *Inceptionism: Going deeper into neural networks*. Online. URL: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah (2018). "Differentiable image parameterizations". In: *Distill* 3.7, e12.

A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski (2017). "Plug & play generative networks: Conditional iterative generation of images in latent space". In: *Proc. CVPR*.

A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune (2016a). "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks". In: *Proc. NeurIPS*.

A. Nguyen, J. Yosinski, and J. Clune (2015). "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proc. CVPR*.

A. Nguyen, J. Yosinski, and J. Clune (2016b). "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks". In: *arXiv preprint arXiv:1602.03616*.

A. P. Norton and Y. Qi (2017). "Adversarial-Playground: A visualization suite showing how adversarial examples fool deep learning". In: *Proc. VizSec*.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter (2020). "Zoom In: An Introduction to Circuits". In: *Distill* 5.3, e00024–001.

C. Olah, A. Mordvintsev, and L. Schubert (2017). "Feature visualization". In: *Distill* 2.11, e7.

C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev (2018). "The building blocks of interpretability". In: *Distill* 3.3, e10.

C. Olah (Oct. 2014). *Visualizing MNIST: An Exploration of Dimensionality Reduction, 2014*. Online. URL: http://colah.github.io/posts/2014-10-Visualizing-MNIST.

C. Olah (Aug. 2015). *Understanding LSTM networks*. Online. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

A. Øygard (July 2015). *Visualizing GoogLeNet Classes*. Online. URL: https://www.auduno.com/2015/07/29/visualizing-googlenet-classes.

G. PAIR (2017). *Facets - Visualizations for ML Datasets*. Online. URL: https://pair-code.github.io/facets.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). "Automatic differentiation in PyTorch". In: *Proc. NeurIPS Workshop*.

M. Patrick, Y. M. Asano, P. Kuznetsova, R. Fong, J. F. Henriques, G. Zweig, and A. Vedaldi (2020). "Multi-modal Self-Supervision from Generalized Data Transformations". In: *arXiv preprint arXiv:2003.04298*.

H. Peng, R. Fong, G. Douaud, C. Beckman, A. Vedaldi, and S. Smith (under review). *Region-Aligned Prediction: Population-level interpretation of deep convolutional network layers in neuroimaging*. Under review.

P. Pérez, M. Gangnet, and A. Blake (2003). "Poisson image editing". In: *ACM SIGGRAPH 2003 Papers*, pp. 313–318.

V. Petsiuk, A. Das, and K. Saenko (2018). "RISE: Randomized Input Sampling for Explanation of Black-box Models". In: *Proc. BMVC*.

Quantstart (2017). *Scalars, Vectors, Matrices, and Tensors - Linear Algebra for Deep Learning (Part 1)*. Online. URL: `https://www.quantstart.com/articles/scalars-vectors-matrices-and-tensors-linear-algebra-for-deep-learning-part-1`.

S.-A. Rebuffi, R. Fong, X. Ji, and A. Vedaldi (2020). "There and Back Again: Revisiting Backpropagation Saliency Methods". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

M. T. Ribeiro, S. Singh, and C. Guestrin (2016). ""Why Should I Trust You?" Explaining the Predictions of Any Classifier". In: *Proc. KDD*.

A. S. Ross, M. C. Hughes, and F. Doshi-Velez (2017). "Right for the right reasons: Training differentiable models by constraining their explanations". In: *arXiv preprint arXiv:1703.03717*.

C. Rudin (2019). "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5, pp. 206–215.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *IJCV*.

M. Sadler and N. Regan (2019). *Game Changer*. New in Chess.

W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K. Müller, eds. (2019). *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer.

W. J. Scheirer, S. E. Anthony, K. Nakayama, and D. D. Cox (2014). "Perceptual annotation: Measuring human vision to improve computer vision". In: *TPAMI* 36.8, pp. 1679–1686.

P. Schulam and S. Saria (2019). "Can you trust this prediction? Auditing pointwise reliability after learning". In: *arXiv preprint arXiv:1901.00403*.

J. Self (1988). *Artificial intelligence and human learning: intelligent computer-aided instruction*. Chapman and Hall London.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2017). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *Proc. ICCV*.

R. R. Selvaraju, S. Lee, Y. Shen, H. Jin, S. Ghosh, L. Heck, D. Batra, and D. Parikh (2019). "Taking a hint: Leveraging explanations to make vision and language models more grounded". In: *Proc. CVPR*.

A. Shrikumar, P. Greenside, and A. Kundaje (2017). "Learning important features through propagating activation differences". In: *Proc. ICML*.

K. Simonyan and A. Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *Proc. ICLR*.

K. Simonyan, A. Vedaldi, and A. Zisserman (2014). "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *Proc. ICLR Workshop*.

K. K. Singh and Y. J. Lee (2017). "Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization". In: *Proc. ICCV*.

M. Skubic and R. A. Volz (2000). "Acquiring robust, force-based assembly skills from human demonstration". In: *IEEE Transactions on Robotics and Automation* 16.6, pp. 772–781.

D. Smilkov, S. Carter, D. Sculley, F. B. Viégas, and M. Wattenberg (2017a). "Direct-manipulation visualization of deep networks". In: *arXiv preprint arXiv:1708.03788*.

D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg (2017b). "Smoothgrad: removing noise by adding noise". In: *arXiv preprint arXiv:1706.03825*.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller (2015). "Striving for simplicity: The all convolutional net". In: *Proc. ICLR*.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *JMLR*.

K. O. Stanley (2007). "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic programming and evolvable machines* 8.2, pp. 131–162.

H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush (2018). "Seq2seq-vis: A visual debugging tool for sequence-to-sequence models". In: *TCVG* 25.1, pp. 353–363.

H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush (2017). "LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks". In: *TCVG* 24.1, pp. 667–676.

P. Sturmfels, S. Lundberg, and S.-I. Lee (2020). "Visualizing the Impact of Feature Attribution Baselines". In: *Distill* 5.1, e22.

M. Sundararajan, A. Taly, and Q. Yan (2017). "Axiomatic attribution for deep networks". In: *Proc. ICML*.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). "Going deeper with convolutions". In: *Proc. CVPR*.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus (2014). "Intriguing properties of neural networks". In: *Proc. ICLR*.

J. Talbot, B. Lee, A. Kapoor, and D. S. Tan (2009). "EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers". In: *Proc. CHI*.

J. Thewlis, H. Bilen, and A. Vedaldi (2017). "Unsupervised Learning of Object Landmarks by Factorized Spatial Embeddings". In: *Proc. ICCV*.

J. Thomas and K. A. Cook (2005). "Illuminating the path: The R&D agenda for visual analytics national visualization and analytics center". In: *National Visualization and Analytics Center*.

J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler (2015). "Efficient object localization using convolutional networks". In: *Proc. CVPR*.

A. Torralba (2017). *DrawNet*. Online. URL: `http://people.csail.mit.edu/torralba/research/drawCNN/drawNet.html`.

M. Tyka (2016). *Class visualization with bilateral filters*. Online. URL: `https://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html`.

D. Ulyanov, A. Vedaldi, and V. S. Lempitsky (2018). "Deep Image Prior". In: *Proc. CVPR*.

J. Wang, Z. Zhang, C. Xie, V. Premachandran, and A. Yuille (2015). "Unsupervised learning of object semantic parts from internal states of CNNs by population encoding". In: *arXiv preprint arXiv:1511.06855*.

X. Wang, A. Shrivastava, and A. Gupta (2017). "A-Fast-RCNN: Hard positive generation via adversary for object detection". In: *Proc. CVPR*.

M. Wattenberg, F. Viégas, and I. Johnson (2016). "How to use t-SNE effectively". In: *Distill* 1.10, e2.

B. Webster (2017). *Now anyone can explore machine learning, no coding required*. Online. URL: `https://www.blog.google/topics/machine-learning/now-anyone-can-explore-machine-learning-no-coding-required`.

D. Wei, B. Zhou, A. Torralba, and W. T. Freeman (n.d.). "Understanding Intra-Class Knowledge Inside CNN". In: *arXiv preprint arXiv:1507.02379* ().

Y. Wei, J. Feng, X. Liang, M.-M. Cheng, Y. Zhao, and S. Yan (2017). "Object region mining with adversarial erasing: A simple classification to semantic segmentation approach". In: *Proc. CVPR*.

J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson (2019). "The What-If Tool: Interactive probing of machine learning models". In: *TCVG* 26.1, pp. 56–65.

K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg (2017). "Visualizing dataflow graphs of deep learning models in tensorflow". In: *TCVG* 24.1, pp. 1–12.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson (2015). "Understanding neural networks through deep visualization". In: *Proc. ICML Deep Learning Workshop*.

M. D. Zeiler and R. Fergus (2014). "Visualizing and understanding convolutional networks". In: *Proc. ECCV*.

J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff (2018). "Top-down neural attention by excitation backprop". In: *IJCV* 126.10, pp. 1084–1102.

Q. Zhang, Y. Nian Wu, and S.-C. Zhu (2018). "Interpretable convolutional neural networks". In: *Proc. CVPR*.

R. Zhang, P. Isola, and A. A. Efros (2016). "Colorful image colorization". In: *Proc. ECCV*.

Z. Zhang, Y. Xie, F. Xing, M. McGough, and L. Yang (2017). "MDNet: A semantically and visually interpretable medical image diagnosis network". In: *Proc. CVPR*.

X. Zhao, Y. Wu, D. L. Lee, and W. Cui (2018). "iForest: Interpreting random forests via visual analytics". In: *TCVG* 25.1, pp. 407–416.

B. Zhou, D. Bau, A. Oliva, and A. Torralba (2018a). "Interpreting deep visual representations via network dissection". In: *TPAMI* 41.9, pp. 2131–2145.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba (2015). "Object detectors emerge in deep scene cnns". In: *Proc. ICLR*.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba (2016a). "Learning deep features for discriminative localization". In: *Proc. CVPR*.

B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva (2016b). "Places: An image database for deep scene understanding". In: *TPAMI*.

B. Zhou, Y. Sun, D. Bau, and A. Torralba (2018b). "Interpretable basis decomposition for visual explanation". In: *Proc. ECCV*.

B. Zhou, Y. Sun, D. Bau, and A. Torralba (2018c). "Revisiting the importance of individual units in cnns via ablation". In: *arXiv preprint arXiv:1806.02891*.

J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros (2016). "Generative Visual Manipulation on the Natural Image Manifold". In: *Proc. ECCV*.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le (2018). "Learning transferable architectures for scalable image recognition". In: *Proc. CVPR*.