Self-Supervised Learning Using Motion and Visualizing Convolutional Neural Networks

Aravindh Mahendran

New College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2018

I dedicate this thesis to my parents, Mahendran Sethuraman and Rukmani Mahendran.

Acknowledgements

I thank my supervisor, Prof. Andrea Vedaldi for his constant support and encouragement without whose guidance this thesis would not be possible. I thank my brother, Siddharth Mahendran, and my parents, Rukmani Mahendran and Sethuraman Mahendran, for constant support and encouragement to keep me going through tough times. I thank my brother, Siddharth Mahendran, and my lab mates for technical discussions, reviewing drafts of several manuscripts and company; including but not limited to: Triantafyllos Afouras, Sam Albanie, Mohsan Alvi, Andrew Brown, Joon Son Chung, Ernesto Coto, Abhishek Dutta, Sébastien Ehrhardt, Ruth Fong, Fatma Guney, Ankush Gupta, João Henriques, Tomas Jakab, Amir Jamaludin, Xu Ji, Sophia Koepke, Vicky Kalogeiton, Marian Longa, Erika Lu, Karel Lenc, Arsha Nagrani, David Novotný, Sylvestre Rebuffi, Christian Rupprecht, Li Shen, James Thewlis, Olivia Wiles, Weidi Xie, Yujie Zhong, Carlos Arteta, Vasileios Belagiannis, Hakan Bilen, Qiong Cao, Yuning Chai, Ken Chatfield, Xi Chen, Mircea Cimpoi, Elliot Crowley, Weilin Huang, Omkar Parkhi, Heba Sailem, Lukas Neumann, Maria Klodt, Oliver Groth, Shiri Avni. I thank Yusuf Aytar for valuable insights regarding the visualizations of our self-supervised model. I also thank my friends in New College, IIIT Hyderabad and Carnegie Mellon University. I thank Anurag Arnab, Tanja Ohlson, Kira L, Matthew Bilyard, Dionysios Kyropoulos and many others whom I shared an accommodation with during my time in Oxford, for making my home away from home comfortable.

I thank Prof. Andrew Zisserman and Prof. Pavan Kumar for helpful feedback during my transfer of status and confirmation of status qualification exams.

I thank Crunchyroll® and Netflix® for all the Japanese anime that kept me going.

I thank my Yoga teacher Mariella De Martini. I thank the New College MCR and its committee for providing a warm and welcoming atmosphere. I also thank the welfare support provided by New College peer supporters and University of Oxford counseling services who have been there when I needed them the most.

I thank Oxford Hub and the Teach Green project and the amazing folks who helped me in these social endeavors.

I gratefully acknowledge funding from grants BP and ERC 677195-IDIU, compute credits from Amazon Cloud Credits for Research program.

Statement of Originality

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, describes my own research.
– Aravindh Mahendran, New College

Abstract

We propose a novel method for learning convolutional image representations without manual supervision. We use motion in the form of optical-flow, to supervise representations of static images. Training a network to predict flow from a single image can be needlessly difficult due to intrinsic ambiguities in this prediction task. We instead propose two simpler learning goals: (a) embed pixels such that the similarity between their embeddings matches that between their optical-flow vectors (CPFS), or (b) segment the image such that optical-flow within segments constitutes coherent motion (S3-CNN). At test time, the learned deep network can be used without access to video or flow information and transferred to various computer vision tasks such as image classification, detection, and segmentation. Our CPFS model achieves state-of-the-art results in self-supervision using motion cues, as demonstrated on standard transfer learning benchmarks.

Despite high transfer learning performance, we feel the need to visualize the representation learned by our self-supervised CPFS model. With that motivation we develop a suite of visualization methods and study several landmark representations, both shallow and deep. These visualizations are based on the concept of "natural pre-image", that is a natural-looking image whose representation has some notable property. We study three such visualizations: inversion, in which the aim is to reconstruct an image from its representation, activation maximization, in which we search for patterns that maximally stimulate a representation component, and caricaturization, in which the visual patterns that a representation detects in an image are exaggerated. We formulate these into a regularized energy-minimization framework and demonstrate its effectiveness. We show that our method can invert HOG features more accurately than recent alternatives while being applicable to CNNs too. We apply these visualization techniques to our self-supervised CPFS model and contrast it with visualizations of a fully supervised AlexNet and a randomly initialized one.

Contents

List of Abbreviations

BP	Back-Propagation
CE	Cross Entropy
Conv	Convolution layer
CNN	Convolutional Neural Network
CPFS	Cross Pixel Flow Similarity
FC	Fully Connected layer
GD	Gradient Descent
HOG	Histogram of Oriented Gradients
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LRN	Local Response Normalization
MLP	Multi-Layer Perceptron
ReLU,RU	Rectified Linear Unit
SIFT	Scale Invariant Feature Transform
S3-CNN	Self-Supervised Segmentation CNN
SGD	Stochastic Gradient Descent
TV	Total Variation
VOC	Visual Object Category
YFCC	Yahoo Flickr Creative Commons

1 Introduction

Convolutional Neural Network (CNN) based models have advanced the state-of-the-art on a wide variety of computer vision tasks such as image classification [58], object detection [112], semantic image segmentation [46], semantic instance segmentation [55], etc. Part of this success is attributed to the model: (a) Convolutions and cross-correlations share parameters across the spatial receptive field effectively treating different locations in the image as data points, (b) Deep Convolutional Neural Networks (Deep CNNs) have the added advantage that deep models can parametrize an exponentially large hypothesis space with a polynomial number of hidden units [8]. However, there is more to the story: Learning such models end-to-end on pixels still requires a large amount of data. Most successful CNN models have been trained on millions of manually annotated images contributing to their success. This thesis views it as a short-coming. Manual annotations are expensive to obtain and will be a bottleneck in the adoption and wide-spread use of large deep CNNs. Furthermore, domains such as medical image analysis and ecosystem studies require experts to annotate data. This further increases the costs, making it very difficult to train or transfer large models into such applications [49].

Self-supervised learning has emerged as a promising approach to address the above problem. The idea is to learn a deep CNN on a proxy task where manual annotations are not required. This model can then be transfered to a target domain / application where only a small number of annotations are available. To take a concrete example, consider the case

of semantic image segmentation. This application requires per-pixel class labels which are expensive to obtain and thus most datasets contain only tens of thousands of annotated images. Training a model on these alone obtains a mere 19.8% mean intersection over union (%mIoU)[1]. To boost performance further, one can pre-train the model using a million image level category labels (ImageNet - ILSVRC2012 [117]) and then fine-tune it for semantic image segmentation. This achieves 48% mIoU establishing that the set of semantic segmentation labels was too small to begin with. In this transfer learning context, the aim for self-supervised learning is to surpass the ImageNet pre-trained model, by pre-training on an appropriate proxy task that does not require any manual annotations. While this has not been achieved yet, considerable progress has been made in the last few years.

1.1 Motion Based Self-Supervision

This thesis contributes two proxy tasks for self-supervised learning of generic **static image representations** in Chapter 3 (CPFS-CE) and Chapter 4 (S3-CNN). These generic features can then be fine-tuned for various target applications such as image classification, object detection, semantic image segmentation etc. Both proxy tasks use motion information as the supervisory signal. To get an intuition for why motion could be useful as a supervisory signal, consider a deep CNN model that is trained to predict, given a single video frame, how the image **could change** over time. Since predicted changes can be verified automatically by looking at the actual video stream, this approach can be used for self-supervision. Furthermore, learning to predict motion may induce a deep network to learn about objects in images. The reason is that objects are a major cause of motion regularity and hence predictability: pixels that belong to the same object are more likely to "move together" than pixels that do not. This is inspired by the **Gestalt principle of common fate** [132] - Things that move together get grouped together.

Besides giving cues about objects, motion has another appealing characteristic compared to other signals for self-supervision. Many other methods are, in fact, based on destroying information in images (e.g. by removing color, scrambling parts) and then tasking a network with undoing such changes. This has the disadvantage of learning the

[1] Details of experiment in section 3.3.5

representation on distorted images (e.g. gray scale). On the other hand, extracting a single frame from a video can be thought of as removing information only along the temporal dimension and allows one to learn the network on undistorted images.

There is, however, a key challenge in using motion for self-supervision: **ambiguity**. Even if the deep network can correctly identify all objects in an image, this is still not enough to predict the specific direction and intensity of the objects' motion in the video, given just a single frame[2]. This ambiguity makes the direct prediction of the appearance of future frames particularly challenging, and overall an overkill if the goal is to learn a general-purpose image representation for image analysis. Instead, the previous most effective method for self-supervision using motion cues [106] is based on first extracting motion tubes from videos (using off-the-shelf optical-flow and motion tube segmentation algorithms) and then training the deep network to predict the resulting per-frame segments rather than motion. Thus they map a complex self-supervision task into one of classic foreground-background segmentation.

While the approach of [106] sidesteps the difficult problem of motion prediction ambiguity, it comes at the cost of pre-processing videos using a complex handcrafted motion segmentation pipeline, which includes many heuristics and tunable parameters. In this thesis, we instead propose two new methods that can ingest cues from motion **directly**. In both cases, motion is represented as optical-flow. Pixel relationships based on optical-flow guide network features to learn about movable objects in the scene. In chapter 4, these relationships are explicitly modeled as a per-pixel non-semantic segmentation. On the other hand, in chapter 3, pixel relationships are implicit in a high dimensional embedding. The latter is more scalable with better transfer learning performance but is much less interpretable.

1.2 Visualizing Convolutional Neural Networks

While the embedding based self-supervised model (CPFS-CE) performs well on transfer learning tasks, it is difficult to get an intuition for why it works and what it is lacking. Models such as these consist of millions of parameters learned from data and their design

[2]Our goal is to learn a static image representation hence only a single frame is used.

Figure 1.1: Four reconstructions of the bottom-right image obtained from the 1,000 dimensional code extracted from the last fully connected layer of the VGG-M CNN [13]. This figure is best viewed in color.

is eminently empirical. Such is the case not just for deep CNNs but also for shallower hand-crafted representations such as HOG [17] and SIFT [88].

With the aim of obtaining a better understanding of representations, we develop a suite of visualization methods to investigate CNNs and other image features. Our methods build on the notion of pre-images, which are images whose representations are notable in some useful sense. For example, their representations may match a target or maximize the activation of a representation component etc. We note that not all pre-images are equally interesting; instead, more meaningful results can be obtained by restricting pre-images to the set of natural images. We call these as "natural pre-images". This is particularly true in the study of discriminative models such as CNNs that are essentially "unspecified" outside the domain of natural images that were used to train them. While capturing the concept of natural images in an algorithm is difficult in practice, we propose to use simple natural image priors as a proxy. We formulate this approach as a regularized energy minimization framework.

All our visualization methods are based on the common idea of seeking natural pre-images whose representations are notable in some useful sense. We propose a unified formulation and algorithm to compute them (section 5.2). Within this framework, we

explore three particular types of visualizations. In the first type, called **inversion**, we compute the "inverse" of a representation (fig. 1.1). We do so by modeling it as a function $\Phi_0 = \Phi(\mathbf{x}_0)$ of the image $\mathbf{x}_0$. Then, we attempt to recover the image from the information contained only in the code Φ_0. Notably, most representations Φ are *not* invertible functions; for example, a representation that is *invariant* to nuisance factors such as viewpoint and illumination removes this information from the image. Our aim is to characterize this loss of information by studying the equivalence class of images $\mathbf{x}^*$ that share the same representation $\Phi(\mathbf{x}^*) = \Phi_0$.

In **activation maximization**, the second visualization type, we look for an image $\mathbf{x}^*$ that maximally excites a certain component $[\Phi(\mathbf{x})]_i$ of the representation. The resulting image is representative of the visual stimuli that are selected by that component and helps understand its "meaning" or function. This type of visualization is sometimes referred to as "deep dream" [93] as it can be interpreted as the result of the representation "imagining" a concept.

In our third and last visualization type, which we refer to as **caricaturization**, we modify an initial image $\mathbf{x}_0$ to exaggerate any pattern that excites the representation $\Phi(\mathbf{x}_0)$. Differently from activation maximization, this visualization method emphasizes the meaning of combinations of representation components that are active together.

Chapter 6 applies this suite of visualization methods to the motion based self-supervised model (CPFS-CE) of chapter 3. It reveals structure contrasting against the structureless pre-images obtained from a randomly initialized network. We also observe a "human face neuron" in the activation maximization output. These experiments also highlight a semantic gap in the representation when compared to the ILSVRC-2012 pretrained AlexNet model [76]. We quantify this semantic gap in terms of the number of interpretable neurons using the Network Dissection method [7]. We also considered using gradient based visualization methods - Network Saliency [121], Guided Backprop [126] and DeConvNets [156], and found that they are not suitable for visualizing individual neurons. Chapter 7 therefore presents a critique and analysis of these methods. It observes that (a) all three methods do *not* visualize individual neurons in deeper layers, (b) Network Saliency and Guided Backprop are selective of foreground objects

and (c) Guided Backprop and DeConvNets show structure in the visualization output. Teasing apart the operations that constitute these methods, we find a notion of fourier phase information in the network which largely influences the visualization of deeper neurons when using these methods.

1.3 Contributions

The core contributions of this thesis are:

- Two methods (CPFS-CE and S3-CNN) for self-supervised learning of generic static image representations using motion as the supervisory signal.
- A suite of visualization methods based on the use of natural image priors - inversion, neuron maximization and caricaturization. We use this suite to visualize standard feedforward CNNs - AlexNet, VGG-M, VGG-VD-16, and a couple of classical computer vision features - HOG, DSIFT.
- Visualizations of our motion based self-supervised model (CPFS-CE). Contrasting between the pre-images of our model, those of a fully-supervised one, and a randomly initialized one reveals more interesting properties about these models and the visualizations themselves.
- A critique and analysis of gradient based network visualization methods - Network Saliency, Guided Backprop and DeConvNets.

1.4 Publications Related to This Thesis

Several chapters are based on the following publications/conference proceedings:

Chapter 3 Aravindh Mahendran, James Thewlis, and Andrea Vedaldi. Cross Pixel Optical-Flow Similarity for Self-Supervised Learning. In *Proceedings of the Asian Conference on Computer Vision*, 2018 (To appear)

Chapter 4 Aravindh Mahendran, James Thewlis, and Andrea Vedaldi. Self-Supervised Segmentation by Grouping Optical-Flow. In *Proceedings of the European Conference on Computer Vision Workshop*, 2018 (To appear)

Chapter 5 Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp 5188-5196, 2015.

Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, December 2016, Volume 120, Issue 3, pp 233–255, 2016.

Chapter 7 Aravindh Mahendran and Andrea Vedaldi. Salient Deconvolutional Networks. In *Proceedings of the European Conference on Computer Vision*, pp 120-135, 2016.

2
Literature Survey

In this chapter we review various prior works that relate to the general topics of this thesis. Past publications specific to individual chapters are discussed within them.

2.1 Self-Supervised Learning

This thesis contributes two methods for self-supervised learning of generic static image representations. We discuss related methods for training generic features for image understanding, as opposed to methods with specific goals such as learning object keypoints. We group these methods according to the supervision cues they use.

Video/Motion Based: Long-term short-term memory based recurrent neural networks can be trained to predict future frames in a video [127]. This requires the network to understand image dynamics and extrapolate those dynamics into the future. However, since several frames are observed simultaneously, these methods may learn something akin to a tracker, with limited abstraction. On the other hand, we learn to predict properties of optical-flow from a **single input image**, thus learning a static image representation rather than a dynamic one. Recently video colorization was used to learn unsupervised tracking [138]. Closely related to our work is the use of *video segmentation* by [106]. They use an off-the-shelf video segmentation method [34] to construct a foreground-background segmentation dataset in an unsupervised manner. A CNN trained on this

proxy task transfers well when fine-tuned for object recognition and detection. We differ from them in that we do not require a sophisticated pre-existing pipeline to extract video segments, but use optical-flow directly. Also closely related to us is the work of [2]. They train a Siamese style CNN to predict the transformation between two images. The individual base networks in their Siamese architecture share weights and can be used as feature extractors for single images at test time. This late fusion strategy forces the learning of abstractions, but our **no-fusion approach** pushes the model even further to learn better features. The polar opposite of this work, is to do early fusion by concatenating two frames as in FlowNet [27]. This was used as a pre-training strategy by [39] to learn representations for **pairs of frames**. This is different from our objective as we aim to learn a **static image representation**. This difference becomes clearer when looking at the evaluation. While we evaluate on image classification, detection, and segmentation; [39] evaluate on dynamic scene and action recognition. [22] use predictive coding for self-supervised disentanglement of pose and content. They encourage content features to be time invariant. Pose features are adversarially learned to not encode content.

Temporal context is a powerful signal. [92, 147, 80] learn to predict the correct ordering of frames. [36] learn features to pick-out the video with incorrect ordering of frames. [62] exploit both temporal and spatial co-occurrence statistics to learn visual groups. [64] extend slow feature analysis using higher order temporal coherence. [144] track patches in a video to supervise their embedding via a triplet loss while [40] do the same but for spatio-temporally matched region proposals. Temporal context is applied in the imitation learning setting by Time Contrastive Networks [120].

Videos contain more than just temporal information. Some methods exploit audio channels by predicting audio from video frames [20, 105]. [3] train a two stream architecture to classify whether an image and sound clip correspond. Temporal information is coupled with ego-motion in [65, 63]. [145] use videos along with spatial context pre-training [23] to construct an image graph. Transitivity in the graph is exploited to learn representations with suitable invariances.

Colorization: [77, 78, 161] predict colour information given greyscale input and show competitive pre-training performance. A generalization to arbitrary pairs of modalities was proposed in [162].

Spatial Context: [107] solve the in-painting problem, where a network is tasked with filling-in partially deleted parts of an image. [23] predict the relative position of two patches extracted from an image. In a similar spirit, [98, 100] solve a jigsaw puzzle problem. The latest iteration on context prediction by [94] obtains state-of-the-art results on several benchmarks.

Adversarial/Generative/Unsupervised: A Bi-directional Generative Adversarial Network (BiGAN) based pretrained model [25] showed competitive transfer learning performance on various recognition benchmarks. [67] adversarially learn to generate and spot defects. Their model trains a discriminator that distinguishes between real and corrupt images. Features from this discriminator can be transfered to various target applications. [113] obtain self-supervision from synthetic data and adapt their model to the domain of real images by training against a discriminator. [10] predict noise-as-targets via an assignment matrix which is optimized on-line. Their approach is domain agnostic. More generally, generative unsupervised layer-wise pre-training was extensively used in deep learning before AlexNet [76]. A review of these and more recent unsupervised generative models is beyond the scope of this thesis.

Transformations/Geometry: [28] create surrogate classes by applying a set of transformations to each image and learn to become invariant to them. [44] do the opposite and try to estimate the transformation (just one of four rotations in their case) given the transformed image. Crop and concatenate transformations are implicit in the 'learning by counting' method of [99]. [101] use correspondences obtained from synthetic warps to learn a dense image representation. Lastly, our optical-flow classification baseline is based on the work of [6]. They learn a sparse hypercolumn model to predict surface normals from a single image and use this as a pretraining strategy. Our baseline flow classification model is the same but with AlexNet for discretized optical-flow.

Others: A combination of self-supervision approaches was explored by [24]. They report that a combination of proxy tasks yields features with better transfer learning performance than each task individually. [159] propose a mix-and-match tuning strategy as a precursor to fine-tuning on the target domain. Their approach can be applied to any pre-trained model and achieves impressive results for PASCAL VOC 2012 semantic segmentation. Another widely-applicable trick that helps in transfer learning is the re-balancing method of [74]. They compute network activations on a set of images which are used to re-scale filter weights. This step is usually done post pre-training and before fine-tuning. [100] cluster features from a pre-trained network to generate pseudo labels, which allows for knowledge distillation from larger networks into smaller ones.

2.2 Visualizing Convolutional Neural Networks

This thesis contributes a suite of visualization methods which are based on "natural pre-images". Here pre-image is an image that is notable in some sense with respect to a representation. It could be an image that matches a target representation or maximizes its components etc. Natural pre-images are pre-images which look natural.

Natural pre-images: Naturalness is achieved in our context using simple regularizers. Our most important regularizer is the quadratic norm of the reconstructed image (section 5.2.2). The visual quality of pre-images can be further improved by introducing complementary regularization methods. Google's "inceptionism" [93], for example, contributed the idea of regularization through jittering: they shift the pre-image randomly during optimization, resulting in sharper and more vivid reconstructions. The work of Yosinksi *et al.* [154] used yet another regularizer: they applied Gaussian blurring and clipped pixels that have small values or that have a small effect on activating components in a CNN representation, to zero. Although the idea of visualizing representations using pre-images has been investigated in connection with neural networks since the work of Linden *et al.* [83], we are the first to use natural image priors in the analysis of modern deep CNNs.

Methods for finding pre-images: Simonyan *et al.* [121] optimized pre-images, starting from random noise and by means of back-propagation and gradient descent, to maximize the response of individual filters in the last layer of a deep CNN. We also use backpropagation for our methods but with a variant of gradient descent based on Adagrad [29]. Related energy-minimization frameworks were later adopted by [93, 154]. Prior to that, very similar methods were applied to early neural networks in [150, 83, 81, 89], using gradient descent or optimization strategies based on sampling.

Several pre-image methods alternative to energy minimization have been explored as well. Nguyen *et al.* [96] used genetic programming to generate images that maximize the response of selected neurons in the very last layer of a modern CNN, corresponding to an image classifier. Vondrick *et al.* [137] learned a paired dictionary that, given a HOG-encoded [17] image patch, inverts it to produce a pre-image. Weinzaepfel *et al.* [148] reconstructed an image from SIFT [88] features using a large vocabulary of patches to invert individual detections and blended the results using Laplace (harmonic) interpolation. Earlier works [68, 134] focussed on inverting networks in the context of dynamical systems.

The DeConvNet method of Zeiler and Fergus [156] "transposes" CNNs to find which image patches are responsible for certain neural activations. While this transposition operation applied to CNNs is somewhat heuristic, Simonyan *et al.* [121] suggested that it approximates the derivative of the CNN and that, thereby, DeConvNet is analogous to one step of the backpropagation algorithm used in their energy minimization framework. A significant difference from our work is that in DeConvNet the authors transfer the pattern of activations of max-pooling layers from the direct CNN evaluation to the transposed one, therefore copying rather than inferring this geometric information during reconstruction. We observe that this geometry dominates the output which no longer visualizes the input neuron for deep layers.

A related line of work [26, 9] is to learn a second neural network to act as the inverse of the original one. This is difficult because the inverse is usually not unique. Therefore, these methods may regress an "average pre-image" conditioned on the target representation, which may not be as effective as sampling the pre-image if the goal is

to characterize representation ambiguities. One advantage of these methods is that they can be significantly faster than energy minimization.

Finally, the vast family of auto-encoder architectures [56] train networks together with their inverses as a form of auto-supervision. We are, however, interested in visualizing discriminatively trained CNNs and self-supervised models based on feed-forward architectures.

Types of visualizations using pre-images: Pre-images can be used to generate a large variety of complementary visualizations, many of which have been applied to a variety of representations in prior work.

The idea of **inverting representations** in order to recover an image from its encoding was used to study SIFT in the work of [148], Local Binary Descriptors by d'Angelo *et al.* [18, 19], HOG in [137] and bag of visual words descriptors in [72]. We also address the inversion problem for HOG, SIFT, and recent CNNs; our method differs significantly from the ones above as it addresses many different representations using the same energy minimization framework and optimization algorithm. In comparison to existing inversion techniques for dense shallow representations such as HOG [137], it is also shown to achieve superior results, both quantitatively and qualitatively.

The first to apply **activation maximization** to recent CNNs such as AlexNet [76] was the work of Simonyan *et al.* [121], where this technique was used to maximize the response of neural activations in the last layer of a deep CNN. Since these responses are learned to correspond to specific object classes, this produces versions of the object as conceptualized by the CNN, sometimes called "deep dreams" [93]. Recently, [93] has generated similar visualizations for their inception network and Yosinksi *et al.* [154] have applied activation maximization to visualize not only the last layers of a CNN, but also intermediate representation components. Related extensive component-specific visualizations were conducted in [156], by searching over a database of patches to find one that maximally excites a neuron. The idea of neuron maximization dates back to at least [30], which introduced activation maximization to visualize deep networks learned from the MNIST digit dataset. It was applied to a large scale multi-layer perceptron

in [79] where they visualized a face-like neuron in their unsupervised network. We also observe a face neuron in our self-supervised model (CPFS-CE).

The first version of **caricaturization** was explored in [121] to maximize image features corresponding to a particular object class, although this was ultimately used to generate saliency maps rather than an image. The authors of [93] extensively explored caricaturization in their "inceptionism" research with two remarkable results. The first was to show which visual structures are captured at different levels in a deep CNN. The second was to show that CNNs can be used to generate aesthetically pleasing images.

In addition to these three broad visualization categories, there are several others which are more specific. In DeConvNet [156] for example, visualizations are obtained by activation maximization. They search in a large dataset for an image that causes a given representation component to activate maximally. The network is then evaluated feed-forward and the location of the max-pooling activations is recorded. Combined with the transposed "deconvolutional network", this information is used to generate crisp visualizations of the excited neural paths. However, this differs from both inversion and activation maximization in that it uses information beyond that contained in the representation output itself.

Fooling pre-images, style transfer and texture synthesis works discussed below are other types of visualizations that do not fit into the three buckets mentioned above. Some of these are used more as image synthesis methods rather than visualization techniques.

Activation statistics: Representations can be used to compute *statistics that describe a class of images*. This idea is rooted in the seminal work of Julesz [71] that used the statistics of simple filters to describe **visual textures**. Julesz' ideas were framed probabilistically by Zhu and Mumford [167] and their generation-by-sampling framework was later approximated by Portilla and Simoncelli [110] as a pre-image problem which can be seen as a special case of the inversion method discussed in this thesis. More recently, Gatys *et al.* [41] showed that the results of Portilla and Simoncelli [110] can be dramatically improved by replacing their wavelet-based statistics with the empirical correlation between deep feature channels in the convolutional layers of CNNs.

Gatys *et al.* further extended their work in [42] with the idea of **style transfer**. Here a pre-image is found that simultaneously (1) reproduces the deep features of a reference "content image" (just like the inversion technique explored by us) while at the same time (2) reproducing the correlation statistics of shallower features of a second reference "style image", treated as a source of texture information. This can be interpreted naturally in our framework as a 'visualization by inversion' where the natural image prior is implemented by "copying" the style of a visual texture. For understanding representations, such a technique can be used to encourage the generation of very different images that share a common deep feature representation, and that therefore may reveal interesting invariance properties of the representation.

Finally, another difference between the work of Gatys *et al.* [41, 42] and the analysis in this thesis is that they transfer information from several layers of the CNN simultaneously, whereas we focus on individual layers, or even single feature components. Thus the two approaches are complementary. In their case, there is no need to add an explicit natural image prior as we do, as this information is incorporated in the low-level CNN statistics that they import in style/texture transfer. As shown in our experiments, a naturalness prior is however important when the goal is to visualize deep features without biasing the reconstructions using this shallower information.

Fooling representations: A line of research related to visualization by pre-images is that of "fooling representations". Here the goal is to generate images, that a representation assigns to a particular category despite having distinctly incompatible semantics. Some of these methods look for *adversarial perturbations* of a source image. For instance, Tatu *et al.* [129] show that it is possible to make any two images look nearly identical in SIFT space up to the injection of adversarial noise in the data. The complementary effect was demonstrated for CNNs by Szegedy *et al.* [128], where an imperceptible amount of adversarial noise was shown to change the predicted class of an image to any desired class. The latter observation was confirmed and extended by [96]. The instability of representations appear in contradiction with our results from chapter 5 and [148, 137]. These show that HOG, SIFT, and early layers of CNNs are largely invertible. This apparent inconsistency may be resolved by noting that [129, 128, 96] require the injection

of adversarial noise which is very unlikely to occur in natural images. It is not unlikely that enforcing representations to be sufficiently regular would avoid the issue.

The work by [96] proposes a second method to generate images that fool representations. They use genetic programming to create, using a sequence of editing operations, an image that is classified as any desired class by the CNN, while not looking like an instance of any class. The CNN does not have a background class that could be used to reject such images; nonetheless the result is remarkable.

3

Cross Pixel Optical-Flow Similarity for Self-Supervised Learning

3.1 Introduction

This thesis contributes two self-supervised methods for learning static image representations using motion cues. The first of these, discussed in this chapter and illustrated in fig. 3.1, is based on a new cross pixel flow similarity (CPFS) framework. This model is inspired by a line of research on predicting the future from a single frame [140, 141, 139, 151, 142, 152]. As discussed in chapter 1, the key challenge with this self-supervision proxy task is that specific details about the motion, such as its direction and velocity, are usually difficult if not impossible to predict from a single frame. This difficulty is, however, essential to the learning of useful image abstractions as it forces the model to use objects rather than pixel tracks as a cue to predict the future. We address this ambiguity in two ways, forming the core of our technical contributions here. First, we learn to embed pixels into vectors that cluster together when the model believes that the corresponding pixels **are likely to move together**. This is obtained by encouraging the inner product of the learned pixel embeddings to correlate with the similarity between their corresponding optical-flow vectors. This does not require the model to explicitly estimate specific motion directions or velocities. However, this is still not sufficient to address the ambiguity completely; in fact, while different objects

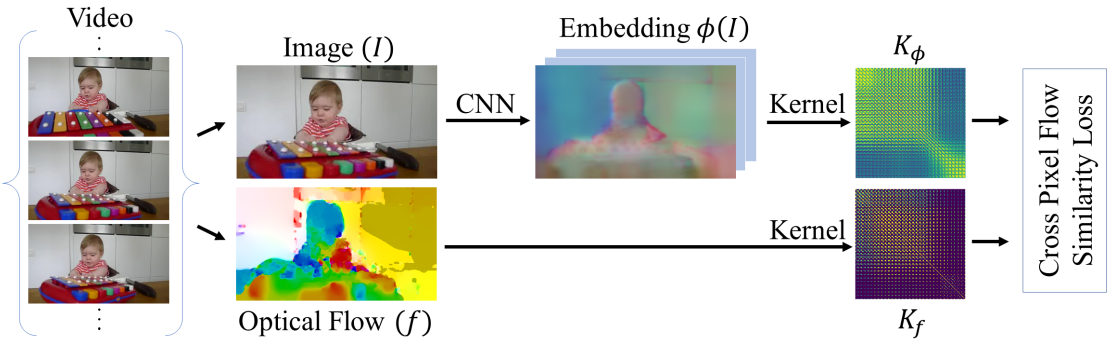


Figure 3.1: We propose a novel method to exploit motion information represented as optical-flow, to supervise the learning of deep CNNs. We learn a network that predicts per-pixel embeddings $\phi(I)$ such that the kernel computed over these embeddings (K_ϕ) is similar to that over corresponding optical-flow vectors (K_f). This allows the network to learn from motion cues while avoiding the inherent ambiguity of motion prediction from a single frame.

may be **able** to move independently, they **may not do so** all the time. For example, objects are often stationary so their velocities are all zero. Then they are all grouped together in optical-flow space. We tackle this second challenge by exploring alternative loss functions within the CPFS framework. Our hope is that some of these loss functions may be more robust than simply using naïve correlation.

Other than ambiguity in motion prediction given a single image, motion as a signal for self-supervision has two issues. (A) Texture less regions in the image cannot be unambiguously matched across two video frames. Thus optical-flow or for that matter any motion signal cannot be estimated in these parts of the image. A simple way to overcome this issue is to only consider regions which can be matched unambiguously. We do not use this trick and rely entirely on smoothness priors used by flow estimation methods to fill-in for texture less regions. (B) Several parts of the image do not move. Thus a notion of objectness relying entirely on the independent motion of objects in the scene can be biased to conclude that there are no distinct objects in the scene. Our method overcomes this because it, roughly speaking, matches against the expectation over possible optical-flows conditioned on the single input frame. So given enough video data all movable objects might be captured in this average.

In section 3.3 we extensively validate our model against other self-supervised learning approaches. First, we show that our approach works as well or better than [106], establishing a new state-of-the-art method for self-supervision using motion cues. Second,

to put this into context, we also compare our results to all recent approaches for self-supervision that use cues other than motion. In this case, we show that our approach, via transfer learning, achieves state-of-the-art performance in semantic image segmentation [1]. We lag behind in classification and detection benchmarks. We also observe that our alternative loss function performs about the same as naïve kernel alignment, suggesting that there is room for improvement on that front.

The overall conclusion (section 3.4) is that our method significantly simplifies leveraging motion cues for self-supervision and does so better than existing alternatives for this modality; it is also competitive with self-supervision methods that use other cues, for the task of semantic image segmentation, making motion a sensible choice for self-supervision by itself or in combination with other cues [24].

3.2 Method

In this section, we describe our novel method, illustrated in fig. 3.1, for self-training deep neural networks via direct ingestion of optical-flow. Once learned, the resulting image representation can be used for classification, detection, segmentation and other tasks with minimal supervision.

Our goal is to learn the parameters of a neural network that maps a single image or frame to a field of pixel embeddings, one for each pixel. Let $\Omega \subset \mathbb{R}^2$ be the set of pixels, then $I : \Omega \to \mathbb{R}^3$ is an image. Our CNN is the per-pixel mapping $\phi(I, p|\Theta) \in \mathbb{R}^D$ producing D dimensional embeddings corresponding to pixel $p \in \Omega$.

In order to learn this CNN, we require the **similarity** between **pairs** of embedding vectors to align with the similarity between the corresponding flow vectors. This is sufficient to capture the idea that things that move together should be grouped together, popularly known as the **Gestalt principle of common fate** [132].

Formally, let $p, q \in \Omega$ be an arbitrary pair of pixels. Their CNN embedding vectors are $\phi(I, p|\Theta), \phi(I, q|\Theta) \in \mathbb{R}^D$. Their corresponding optical-flow vectors are $f_p, f_q \in \mathbb{R}^2$.

[1] At the time this work was submitted to ACCV-2018

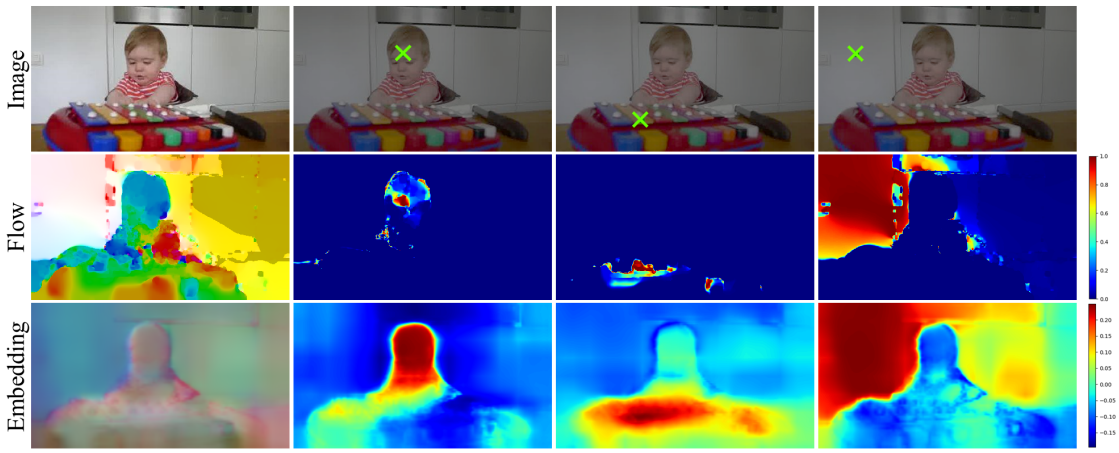


Figure 3.2: Visualization of flow and embedding kernels, in the second and third rows respectively. For three pixels p, we plot the row $K_*(p, \cdot)$ reshaped into an image, showing which pixels go together from the kernel's perspective. Note the localized nature of the flow kernel which is obtained by setting a low bandwidth for the RBF kernel. $\sigma^2 = 0.0036$ in this example. In the first column, optical-flow and embeddings (after a random $16D \to 3D$ projection) are visualized as color images.

We match their kernels:

$$\forall p, q \in \Omega : \quad K_\phi\Big(\phi(I, p|\Theta), \phi(I, q|\Theta)\Big) \approxeq K_f(f_p, f_q) \tag{3.1}$$

where $K_\phi : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, $K_f : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ are kernels that measure the similarity of the CNN embeddings and flow vectors, respectively.

In this formulation, in addition to the choice of CNN architecture ϕ, the key design decisions are the choice of kernels K_ϕ, K_f and how to translate constraint eq. (3.1) into a loss function. The rest of the section discusses these choices.

3.2.1 Kernels

In order to compare CNN embedding vectors and flow vectors, we choose the (scaled) cosine similarity kernel and the Gaussian/RBF kernel respectively. Using the shorthand notation $\phi_p = \phi(I, p|\Theta)$ for readability, these are:

$$K_\phi(\phi_p, \phi_q) := \frac{1}{4} \frac{\phi_p^T \phi_q}{\|\phi_p\|_2 \|\phi_q\|_2}, \qquad K_f(f_p, f_q) := \exp\left(-\frac{\|f_p - f_q\|_2^2}{2\sigma^2}\right). \tag{3.2}$$

Note that these kernels, when restricted to the set of pixels Ω, are matrices of size $|\Omega| \times |\Omega|$. Each row or column of this matrix can be thought of as a heatmap capturing the similarity

of a given pixel with respect to all other pixels and thus can be visualized as an image. We present such visualizations for both of our kernels in fig. 3.2.

We use the Gaussian kernel for the flow vectors as this is consistent with the Euclidean interpretation of optical-flow as a displacement. Reducing kernel bandwidth (σ) would result in a localized kernel that pushes our embeddings to distinguish between different movable objects. In some experiments, the value of σ is learned along with the weights of the CNN in the optimization. This localized kernel, with learned $\sigma^2 = 0.0036$, is shown in the second row of fig. 3.2.

We use the cosine kernel for the learned embedding as the CNN effectively computes a *kernel feature map*, so that in principle it can approximate any kernel via the inner product. However, note that the expression normalizes vectors in L^2 norm, so that this inner product is the cosine of the angle between embedding vectors. This normalization is key as it guarantees that K_Φ is maximum when the embeddings being compared are identical (the Gaussian kernel does so automatically).

3.2.2 Cross Pixel Optical-Flow Similarity Loss Function

The constraint in eq. (3.1) requires kernels K_ϕ and K_f to be similar. We experiment with three loss functions for this task - kernel target alignment, cross-entropy, and cross-entropy reversed.

Kernel Target Alignment (KTA): KTA [15] is a conventional metric to measure the similarity between kernels. KTA for two kernel matrices $K \in \mathbb{R}^{D\times D}$, $K' \in \mathbb{R}^{D\times D}$, is given by

$$\mathcal{L}_{KTA}(K, K') = \sum_{pq} K_{pq} K'_{pq} / \sqrt{\sum_{pq} K_{pq}^2 \sum_{pq} {K'_{pq}}^2} \tag{3.3}$$

Cross-Entropy (CE): Our second loss function treats pixels as classes and kernel values as logits of a distribution over pixels. The cross entropy of these two distributions measures the distance between them. We compute this loss in two steps. First, we re-normalize each column $K_*(\cdot, q)$ of each kernel matrix into a probability distribution $S_*(\cdot, q)$ [2]. $S_f(\cdot, q)$ describes which image pixels p are likely to belong to the same

[2] K_* denotes both K_f and K_ϕ, S_* denotes both S_f and S_ϕ

segment as pixel q, according to optical-flow. $S_\phi(\cdot, q)$ describes the same but from the CNN embedding's perspective. These distributions, arising from CNN and optical-flow kernels, are compared by using cross entropy, summed over columns:

$$\mathcal{L}_{CE}(\Theta) = -\sum_q \sum_p S_f(p,q) \log S_\phi(p,q). \tag{3.4}$$

Normalization uses the softmax operator. We reduce the contribution of diagonal terms in the kernel matrix before this normalization because each pixel is trivially similar to itself and would skew the softmax. Formally, for optical-flow we have:

$$S_f(p,q) = \begin{cases} 1/\left(\sum_{q' \neq p} \exp(K_f(p,q')) + 1\right), & \text{if } p = q, \\ \exp(K_f(p,q))/\left(\sum_{q' \neq p} \exp(K_f(p,q')) + 1\right), & \text{if } p \neq q. \end{cases} \tag{3.5}$$

Similarly, for the CNN embedding we have:

$$S_\phi(p,q) = \begin{cases} 1/\left(\sum_{q' \neq p} \exp(K_\phi(p,q')) + 1\right), & \text{if } p = q, \\ \exp(K_\phi(p,q))/\left(\sum_{q' \neq p} \exp(K_\phi(p,q')) + 1\right), & \text{if } p \neq q. \end{cases} \tag{3.6}$$

Cross-Entropy Reversed (CE-rev): Note that the particular ordering of distributions inside the cross entropy loss of eq. (3.4) treats the distribution induced by the optical-flow kernel (S_f) as ground truth. The embedding is tasked with inducing a kernel such that its corresponding distribution S_ϕ matches S_f. As an ablation study we also experiment with the order of distributions reversed. In other words we use,

$$\mathcal{L}_{CE-rev}(\Theta) = -\sum_q \sum_p S_\phi(p,q) \log S_f(p,q). \tag{3.7}$$

The intuition here is as follows: For a given pixel p, the distribution $S_\phi(\cdot, q)$ must be a delta distribution around $q' = \operatorname{argmax} S_f(\cdot, q)$. This is the natural effect of a flipped cross entropy loss. This delta distribution can be best approximated by aligning the two embeddings $\phi_p \cong \phi_{q'}$ and making all others anti-correlated $\phi_p \cong -\phi_q \ \forall \ q \neq q'$. Note however that this degenerate solution forces all $\phi_q, \phi_{q''}$ such that $q, q'' \neq q'$ to align as well. This coincidental alignment would, in general, significantly increase the loss function. Thus the embedding is forced to induce a non degenerate distribution S_ϕ. We consider it interesting to experiment with this loss.

Thus we have three cross pixel flow similarity losses - Kernel Target Alignment (CPFS-KTA), Cross Entropy (CPFS-CE) and Cross Entropy reversed (CPFS-CE-rev).

3.2.3 CNN Embedding Function

Lastly, we discuss the architecture of the CNN function ϕ itself. It maps the image to a per-pixel embedding. Recall that $\forall\ p \in \Omega,\ \phi(I, p|\Theta) \in \mathbb{R}^D$. We design the embedding CNN as a hypercolumn head [53] over a conventional CNN backbone such as AlexNet. The hypercolumn concatenates features from multiple depths so that our embedding can exploit high resolution details normally lost due to max-pooling layers. For training, we use the sparsification trick of [77, 78] and restrict prediction and loss computation to a few randomly sampled pixels in every iteration. This reduces memory consumption and improves training convergence as pixels in the same image are highly correlated and redundant; via sampling we can reduce this correlation and train more efficiently [6].

In more detail, the backbone is a CNN with activations at several layers: $\{\phi_{c_1}(I|\Theta), \cdots,\ \phi_{c_n}(I|\Theta)\} \in \mathbb{R}^{H_1 \times W_1 \times D_1} \times \cdots \times \mathbb{R}^{H_n \times W_n \times D_n}$. We follow [78] and interpolate values for a given pixel location and concatenate them to form a hypercolumn $\phi_H(I, p|\Theta) \in \mathbb{R}^{D_1 + \cdots + D_n}$. The hypercolumn is then projected non-linearly to the desired embedding $\phi(I, p|\Theta) \in \mathbb{R}^D$ using a multi-layer perceptron (MLP). Details of the model architecture are discussed in section 3.3.1.

3.3 Experiments

We extensively assess our approach by demonstrating its effectiveness in learning features that we show as useful for several tasks. In order to make our results comparable to most of the related papers in the literature, we consider an AlexNet [76] backbone and four tasks: classification in ImageNet [117] and classification, detection, and segmentation in PASCAL VOC [32, 33].

3.3.1 Backbone Details

We adapt the AlexNet version used by Pathak *et al.* [106]. The modifications are minor (mostly related to padding) and to make it suitable to attach a hypercolumn head. Sparse hypercolumns are built from the conv1, pool1, conv3, pool5 and fc7 AlexNet activations. Embeddings are generated using a multi-layer perceptron (MLP) with a single hidden

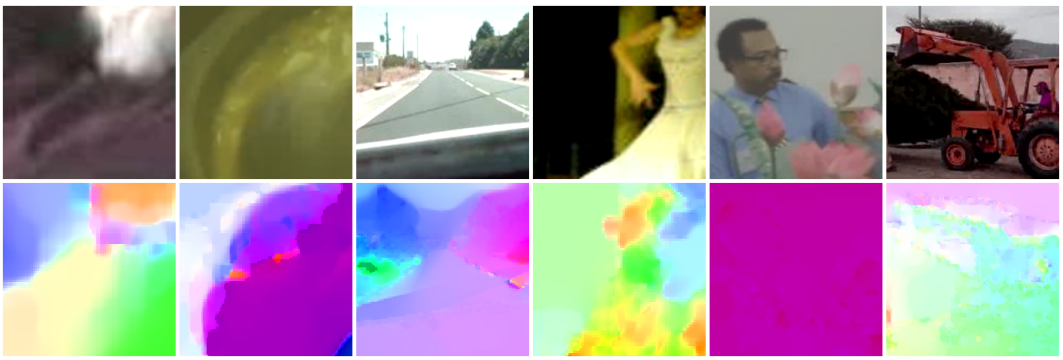


Figure 3.3: Image and optical-flow training pairs post scale-crop-flip data augmentation. The noisy nature of both images and optical-flows illustrate the challenges in using motion as a self-supervision signal. Optical-flow is visualized as a colour image using the toolbox of [11].

layer and are L2-normalized. The embeddings are $D = 16$ dimensional (this number could be improved via cross validation, although this is expensive). The exact model specification, in the caffe text protocol buffer format (.prototxt), is included in appendix A.

3.3.2 Dataset

We train the above AlexNet model on a dataset of RGB-optical-flow image pairs. Inspired by the work of Pathak *et al.* [106], we built a dataset from $\sim 204k$ videos in the YFCC100m dataset [131]. The latter consists of Flickr videos made publicly-available under the creative commons license. We extract 8 random frames from each video and compute optical-flow between those at times t and $t + 5$ using the same (handcrafted) optical-flow method of [106, 84]. Overall, we obtain 1.6M image-flow pairs.[3] Example training sample crops along with optical-flow fields are shown in fig. 3.3. The noisy nature of both the images and optical-flow in such large-scale non-curated video collections makes it all the more challenging for self-supervision.

Optical-flow vectors (f_x, f_y) are normalized logarithmically to lie between $[-1, 1]$ during training, so that occasional large flows do not dominate learning. More precisely,

[3]The dataset occupies more than 1TB of space and does not easily fit in fast memory such as an SSD for training. We addressed this problem by using a fixed point 16bit representation of optical-flow similar to KITTI [43] and storing it as PNG images, with dramatic compression and negligible residual error. The compressed data occupies 431GB.

the normalization is given by:

$$f' = \begin{bmatrix} \text{sign}(f_x) \min\left(1, \frac{\log(|f_x|+1)}{\log(M+1)}\right) \\ \\ \text{sign}(f_y) \min\left(1, \frac{\log(|f_y|+1)}{\log(M+1)}\right) \end{bmatrix} \tag{3.8}$$

where M is a loose upper bound on the flow-magnitude set to 56.0 in our experiments.

Despite the large size of this data and aggressive data augmentation during training, AlexNet overfits on our self-supervision task. We use early stopping to reduce over-fitting by monitoring the loss on a validation set. The validation set consists of 5000 image-flow pairs computed from the YouTube objects dataset [111]. Epic-Flow [114], with initial matches from Deep-Matching [149], was used to compute optical-flow for these frames.

3.3.3 Learning Details

We use the Adam optimizer [73] with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ and initial learning rate set to 10^{-4}. No weight decay is used because it resulted in our model reaching a worse local minima before overfitting started. Pixels are sampled uniformly at random for the sparse hypercolumns. Sampling more pixels gives more information per image but also consumes more memory and is computationally expensive, making each iteration slower. We use 512 pixels per image to balance this trade-off. This reduces memory consumption and allows for a large batch size of 96 frames. Scale, horizontal flip and crop augmentation are applied during training. Color augmentation: shifting the hue by up to 0.1, random contrast between 0.2 and 1.8, random brightness by up to 0.12 (based on $0-1$ normalised colours); is also applied. The network is trained on crops of size 224×224. Parameter-free batch-normalization [60] is used throughout the network; the moving average mean and variance are absorbed into convolution kernels after self-supervised training, so that, for evaluation, AlexNet does not contain batch normalization layers. The full implementation using TensorFlow [1] will be publicly available on GitHub.

3.3.4 Qualitative Results

While our learned pixel embeddings are not meant to be used directly (instead their purpose is to pre-train a neural network parametrization that can be transferred to other

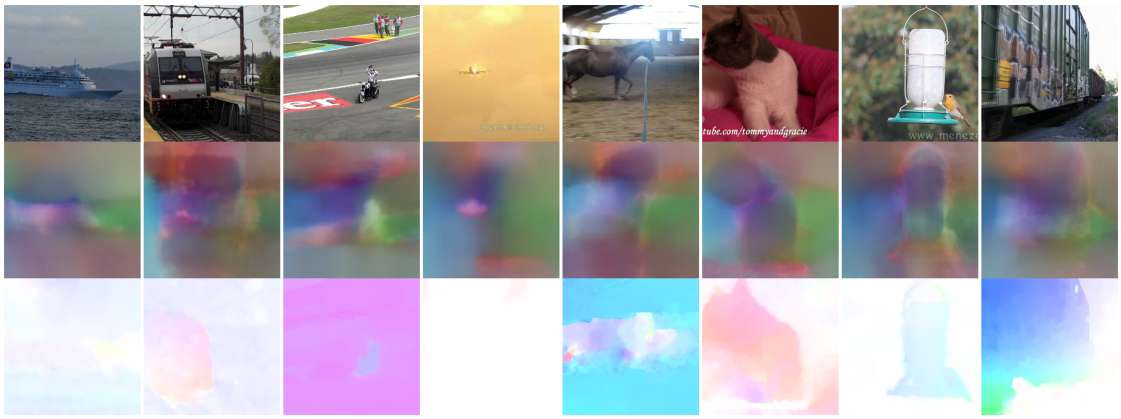


Figure 3.4: Per-pixel embeddings are visualized by randomly projecting them to RGB colors. We show example embeddings generated by our model on frames from the YouTube objects dataset. From top to bottom: The original validation images, RGB-mapped embeddings, and optical-flow fields. Best viewed in color on screen.

tasks), nevertheless it is informative to visualize them. Since embeddings are 16D, we first project them to 3D vectors via random projections and then map the resulting coordinates to RGB space. We visualize embeddings generated by our model, pre-trained using the CPFS-CE loss (Equation (3.4)), on frames from the YouTube objects validation set in fig. 3.4. Note that pixels on a salient foreground object tend to cluster together in the embedding (see, for example, the aircraft in column 4, the motor cyclist in column 3 and the cat in column 6).

3.3.5 Quantitative Results

We follow standard practice in the self-supervised learning community and fine-tune the learned representation on various recognition benchmarks. We evaluate our features by transfer learning on PASCAL VOC 2007 detection and classification [32], PASCAL VOC 2012 segmentation [33], and ILSVRC12 linear probing [161] (in the latter case, the representation is frozen). We provide details on the evaluation protocol next and compare against other self-supervised models with results reported for AlexNet-like-architectures in tables 3.1 and 3.2. Different from other approaches, we did not benefit from the re-balancing trick of [74] and hence we report results without it. This is possibly due to the use of batch normalization layers.

Baseline: Our main hypothesis is that the cross pixel flow similarity matching method, rather than the direct prediction of optical-flow, is more appropriate for exploiting optical-flow as a self-supervisory signal. We validate this hypothesis by comparing against a direct optical-flow prediction baseline, using the same CNN architecture as our method but a different loss function: while we use flow-similarity matching losses, this baseline does a standard per-pixel softmax cross entropy across 16 discrete optical-flow classes, once for each spatial dimension — x and y. To this end, since the flow is normalized in $[-1, 1]$ (eq. (3.8)), this interval is discretized uniformly. Note that direct L^2 regression of flow vectors is also possible, but did not work as well in preliminary experiments. This may be because continuous regression is usually harder for deep networks compared to classification, especially for ambiguous tasks. It was beneficial to use a faster initial learning rate of 0.01.

VOC2007-detection: We finetune our AlexNet backbone end-to-end using the Fast-RCNN model [45] using code from [112] to obtain results for PASCAL VOC 2007 detection [32]. Finetuning is performed for 150k iterations, were the learning rate drops by a 10^{th} every 50k iterations. The initial learning rate is 0.002, weight decay is 5×10^{-4}, train-set is VOC2007-train+val, test-set is VOC2007-test. Following the guidelines of [74], we use multi-scale training and single scale testing. We report mean average precision (mAP) in table 3.1 (col. 5) along with results of other self-supervised learning methods. We achieve the state-of-the-art among methods that use temporal information in videos for self-supervision.

VOC2007 classification: We finetune our pretrained AlexNet to minimize the softmax cross-entropy loss over the PASCAL VOC 2007 *trainval* set for image classification across 20 Pascal classes. The initial learning rate is 10^{-3} and drops by a factor of 2 every 10k iterations for a total of 80k iterations and predictions are averaged over 10 random crops at test time in keeping with [74]. We use the code provided by [78] and report mean average precision (mAP) on VOC2007-test in the fourth column of table 3.1. We achieve state-of-the-art among methods that derive self-supervision from motion cues; in particular, we outperform [106] by a large margin.

	Method	Supervision	[Ref]	Cls.	Detection	Seg.
	Krizhevsky *et al.* [76]	Class Labels	[161]	79.9	56.8	48.0
	Random	-	[107]	53.3	43.4	19.8
Motion cues	Agrawal *et al.* [2]	Egomotion	[25]	63.1	43.9	-
	Jayaraman *et al.* [63]	Egomotion	[63]	-	41.7	-
	Lee *et al.* [80]	Time-order	[80]	63.8	46.9	-
	Misra *et al.* [92]	Time-order	[92]	-	42.4	-
	Pathak *et al.* [106]	Video-seg	[106],Self	61.0	50.2	-
	Wang *et al.* [144]	Track + Rank	[74, 144]	63.1	47.5	-
	CPFS-CE	Optical-flow	Self	64.2	**50.8**	41.4
	CPFS-CE-rev	Optical-flow	Self	63.6	49.9	39.5
	CPFS-KTA	Optical-flow	Self	**65.3**	50.5	**41.5**
	Ours direct cls.	Optical-flow	Self	63.2	46.1	38.8
Other cues	Bojanowski *et al.* [10]	-	[10]	65.3	49.4	-
	Doersch *et al.* [23]	Context	[25]	65.3	51.1	-
	Donahue *et al.* [25]	-	[25]	60.3	46.9	35.2
	Gidaris *et al.* [44]	Rotation	[44]	**73.0**	54.4	39.1
	Krahenbuhl *et al.* [74]	-	[74, 25]	56.6	45.6	32.6
	Larssons *et al.* [78]	Colorization	[78]	65.9	-	38.4
	Mundhenk *et al.* [94]	Context	[94]	69.3	55.2	**40.6**
	Noroozi *et al.* [98]	Jigsaw	[98]	67.6	53.2	37.6
	Noroozi *et al.* [99]	Counting	[99]	67.7	51.4	36.6
	Noroozi *et al.* [100]	Jigsaw++	[100]	69.8	**55.5**	38.1
	Noroozi *et al.* [100]	CC+Jigsaw++	[100]	69.9	55.0	40.0
	Owens *et al.* [105]	Sound	[162, 105]	61.3	44.1	-
	Pathak *et al.* [107]	In-painting	[107]	56.5	44.5	29.7
	Jenni *et al.* [67]	-	[67]	69.8	52.5	38.1
	Zhang *et al.* [161]	Colorization	[161]	65.9	46.9	35.6
	Zhang *et al.* [162]	Split-Brain	[162]	67.1	46.7	36.0
	Zhan *et al.* [159]†	Colorization	[159]	-	-	42.8
	Zhan *et al.* [159]†	Puzzle	[159]	-	-	44.5

Table 3.1: Pascal VOC Comparison for three benchmarks: VOC2007-classification (column 4) %mAP, VOC2007-Detection (Column 5) %mAP and VOC2012-Segmentation (Column 6) %mIoU. The rows are grouped into four blocks (1) The limits of no-supervision and human supervision, (2) motion/video based self-supervision, (3) Our models and the baseline, (4) other self-supervision cues. The third column "[ref]" indicates which publication the reported numbers are borrowed from. Zhan *et al.* (marked †) contribute a different mix-and-match tuning strategy which transfers better to the target domain compared to finetuning. This is orthogonal to the efforts of finding a self-supervision method and is therefore not counted when marking the state-of-the-art in **bold.**

	Method	Supervision	[ref]	C1	C2	C3	C4	C5
	Krizhevsky *et al.* [76]	Class Labels	[162]	19.3	36.3	44.2	48.3	50.5
	Random	-	[162]	11.6	17.1	16.9	16.3	14.1
	Random-rescaled [74]	-	[74]	17.5	23.0	24.5	23.2	20.6
Motion	Pathak *et al.* [106]	Video-seg	Self	**15.8**	23.2	29.0	29.5	25.4
Motion	CPFS-CE	Optical-Flow	Self	14.9	**25.0**	**29.5**	**30.1**	**29.1**
Motion	CPFS-CE-rev	Optical-Flow	Self	15.3	24.8	27.7	27.8	26.3
Motion	CPFS-KTA	Optical-Flow	Self	14.8	24.6	29.2	29.5	28.1
Motion	Ours direct cls.	Optical-Flow	Self	14.0	23.0	26.4	26.7	24.8
Other cues	Doersch *et al.* [23]	Context	[162]	16.2	23.3	30.2	31.7	29.6
Other cues	Gidaris *et al.* [44]	Rotation	[44]	18.8	31.7	**38.7**	38.2	**36.5**
Other cues	Jenni *et al.* [67]	-	[67]	19.5	**33.3**	37.9	**38.9**	34.9
Other cues	Donahue *et al.* [25]	-	[162]	17.7	24.5	31.0	29.9	28.0
Other cues	Mundhenk *et al.* [94]	Context	[94]	**19.6**	31.4	37.0	37.8	33.3
Other cues	Noroozi *et al.* [98]	Jigsaw	[99]	18.2	28.8	34.0	33.9	27.1
Other cues	Noroozi *et al.* [99]	Counting	[99]	18.0	30.6	34.3	32.5	25.7
Other cues	Noroozi *et al.* [100]	Jigsaw++	[100]	18.2	28.7	34.1	33.2	28.0
Other cues	Noroozi *et al.* [100]	CC+Jigsaw++	[100]	18.9	30.5	35.7	35.4	32.2
Other cues	Pathak *et al.* [107]	In-Painting	[162]	14.1	20.7	21.0	19.8	15.5
Other cues	Zhang *et al.* [161]	Colorization	[162]	13.1	24.8	31.0	32.6	31.8
Other cues	Zhang *et al.* [162]	Split-Brain	[162]	17.7	29.3	35.4	35.2	32.8

Table 3.2: ImageNet LSVRC-12 linear probing evaluation. A linear classifier is trained on the (downsampled) activations of each layer in the pretrained model. Top-1 accuracy is reported on ILSVRC-12 validation set. The column "[ref]" indicates which publication the reported numbers are borrowed from. We finetune Pathak *et al.*'s [106] model along with ours as they do not report this benchmark in their paper. C1: Conv1, C2: Conv2, C3: Conv3, C4: Conv4, C5: Conv5

ILSVRC12 linear probing: We follow the protocol and code of [162] to train a linear classifier on activations of our pre-trained network. The activation tensors produced by various convolutional layers (after ReLU) are down-sampled using bilinear interpolation to have roughly 9,000-10,000 elements before being fed into a linear classifier. The CNN parameters are frozen and only the linear classifier weights are trained on the ILSVRC-12 training set. Top-1 classification accuracy is reported on the ILSVRC-12 validation set (table 3.2). We achieve the state-of-the-art among motion based self-supervision methods, except for "conv1" features.

VOC2012 segmentation: We use the two staged fine-tuning approach of [78] who finetune their AlexNet for semantic segmentation using a sparse hypercolumn head instead of the conventional FCN-32s head. We do so because it is a better fit for our sparse hypercolumn pre-training, although the hypercolumn itself is built using different layers (conv1 to conv5 and fc6 to fc7). Thus the MLP predicting the embedding ϕ from hypercolumn features is replaced with a new one before fine-tuning for segmentation. Also, our model has a fully convolutional structure but is pre-trained on a non-convolutional proxy task. We obtain a mere 31.3 %mIoU using FCN-32s.

[78]'s two staged fine-tuning approach involves training the model twice: First train on 90% of the training data while monitoring the per pixel accuracy on the remaining 10%. If the accuracy drops or plateaus, drop the learning rate by a factor of 10 and continue. Next, freeze the learning rate scheme obtained by the first pass and train on the entire training set again. The training data consists of the PASCAL VOC 2012 [33] training set augmented with additional annotations from [52]. Thus the training-validation split has 10582 training images and 1449 validation images respectively. Test results are reported as mean intersection-over-union (mIoU) scores on the PASCAL VOC 2012 validation set (Column 6 of table 3.1). We achieve the state of the art on this benchmark among all self-supervised learning methods, even ones that use other supervisory signals than motion (at the time of submission of this work to ACCV-2018).

Other Architectures: We experimented with a VGG-16 [123] backbone[4] and followed the protocol of [78] to evaluated on VOC2007-classification and VOC12-segmentation. Our CPFS-CE model achieved 76.4% mAP for VOC2007 classification comparable to Larsson *et al.*'s 77.2% [78]; and 51.7% mIoU for VOC2012-segmentation which fell short of [78]'s 56.0%. VGG-16 has many more parameters than AlexNet. We argue that it may benefit from the extra 2.1M images used by [78] which might explain this performance gap.

3.3.6 Discussion

We can take home several messages from these experiments. First, in all cases our approach outperforms the baseline of predicting optical-flow directly. This is true for all three cross pixel flow similarity loss functions. This supports our hypothesis that direct single-frame optical-flow prediction is either too difficult due to its intrinsic ambiguity or a distraction from the goal of learning a powerful representation. It also supports our claim that predicting pairwise flow similarities partially addresses this ambiguity and allows us to learn useful CNN representations from optical-flow.

Second, the cross entropy loss is comparable in performance to kernel target alignment (KTA). We know that KTA aligns kernels uniformly and doing so is still highly ambiguous. Thus there is more room for improvement in the loss function design. Perhaps a probabilistic treatment of kernel matrices may be useful in the future. Also, surprisingly, reversed cross entropy performs well although not as well as the other two loss functions.

Third, our method is the state of the art for self-supervision using motion cues. This is notable as our approach is significantly simpler than the previous state of the art represented by [106]. By ingesting optical-flow information directly, it does not require data pre-processing via a video segmentation algorithm.

Finally, we also observe that all video/motion based methods for self-supervised learning are generally not as effective as methods that use other cues; particularly in image classification benchmarks. However, our approach still sets the overall state of the art for semantic image segmentation suggesting that the learned representation may be

[4]Full model: 'pool 1-5', and 'fc7' (projected to 256 channels using a 1×1 convolution for faster training) constitute the hypercolumn head for pre-training on the dataset (Section 3.3.2).

more suitable for per-pixel applications. Therefore further progress in this area of motion based self-supervision may be possible and is worth seeking. At the same time, authors of [24] find that combinations of different cues may result in the best performance; in this sense, our approach, by significantly simplifying the use of motion cues, can make it much easier to design multi-task networks that can leverage motion together with other complementary methods.

3.4 Summary

We have presented a novel method for self-supervision using motion cues based on cross-pixel optical-flow similarity loss functions. We trained an AlexNet model using this scheme on a large unannotated video data-set.

We established the effectiveness of the resulting representation by transfer learning for several recognition benchmarks. Compared to the previous state of the art motion based method [106], our method works just as well and in some cases noticeably better despite a significant algorithmic simplification. We also outperform all other self-supervision strategies in semantic image segmentation (VOC12). This is reasonable as we train on a per-pixel proxy task on undistorted RGB images and use a hypercolumn model for fine-tuning.

3.5 Statement of Authorship

This chapter is based on joint work with James Thewlis (CDT, Department of Engineering Science, University of Oxford). His individual contributions are detailed below:

1. Prepared a smaller version of YFCC100m-1600k dataset (section 3.3.2) called YFCC100m-150k. The latter is also used in chapter 4.

2. Formulated and implemented the cross entropy based cross pixel flow similarity loss (CPFS-CE) (eq. (3.4)).

3. Developed and engineered the first form of the flow normalization method. Equation (4.10) of chapter 4.

4

Self-Supervised Segmentation by Grouping Things That Flow Together

4.1 Introduction

This chapter presents our second model for self-supervised learning of static image representations using motion cues. It is a variant of the model discussed in chapter 3. The core insight is that, instead of an implicit grouping by a similarity over embeddings as in chapter 3, one can learn a model that **segments the image into regions**. Then the proxy task for self-supervision is to predict segments that have **coherent motion** within them. The key advantage of this approach, over the model in chapter 3, is that it enables the use of more sophisticated measures of coherent motion, such as fitting an affine transformation, or a fundamental matrix. Such transformations capture groupings in optical-flow space that cannot be expressed as pairwise stationary kernels. Another benefit is that explicit segmentation is a more interpretable output.

The proposed model does not observe motion in its input. It is observed only by the loss function. The model is tasked with predicting, by looking at a single image alone, which *groups of pixels are likely to move together*. These often turn out to belong to individual rigid objects/parts. To see this, consider for instance an object translating in the image. It can be segmented in one of two ways: (a) The object is segmented by itself, (b) The object and background are merged together. In the former case, motion within

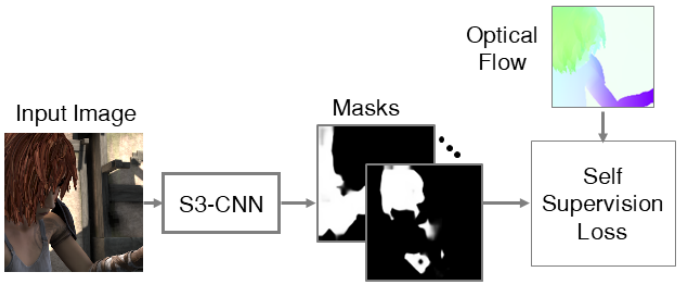


Figure 4.1: S3-CNN framework. We propose to learn a neural network operating on images using temporal information contained in videos as supervision. The self-supervision proxy task is to predict regions that are likely to have "coherent" optical-flow. Flow can be observed by the loss, but not by the neural network. "Coherent" in this context could mean to have affine or rigid motion etc., encouraging the network to learn about object-part-like regions in images.

the segment will be coherent. In the latter case it will be less so. Thus encouraging the segments to have coherent motion is likely to encourage the grouping of objects/object-parts. We call the resulting objective **self-supervised segmentation** (*S3*; fig. 4.1) and the models trained using it S3-CNN.

It is interesting to contrast this method with the obvious alternative of learning to estimate the *conditional distribution* $p(\mathbf{y}|\mathbf{x})$ of the optical-flow $\mathbf{y}$ from a single image $\mathbf{x}$. Such a formulation is often used in proxy tasks such as colorization, where $\mathbf{y}$ could be the color channels of a grayscale image $\mathbf{x}$. In such cases, the conditional distribution may be sufficiently simple to afford direct modeling, for example under the i.i.d. approximation. However, if $\mathbf{y}$ is the optical-flow, since it is usually impossible to uniquely predict the direction of motion given a single frame $\mathbf{x}$, one may be forced to estimate a complex, multi-modal posterior $p(\mathbf{y}|\mathbf{x})$. This was attempted by the flow classification baseline in section 3.3.5 of the previous chapter. In that chapter our CPFS models sidestepped predicting complex posteriors by matching similarity between flows instead of predicting flows directly. In this chapter we predict segments over which the flow is likely to be coherent in some sense, also without predicting the direction or intensity of the flow.

The main strength of this method is that we can now use sophisticated measures of coherent motion that cannot be captured using pairwise stationary kernels. We learn S3-CNN networks using three coherency measures - affine model fit, rigid model fit and low entropy flow histograms (section 4.3). The affine and rigid models work well for

synthetic data with ground truth optical-flow. The low-entropy-flow-histogram model works well on real world video data with computed optical-flow. In all cases we visualize segments predicted by our networks. We also show quantitative results by transferring the learned model for recognition tasks (section 4.4), where we come close to matching the performance of colorization networks on VOC-2007 classification. Unfortunately this model does not scale well and the learning dynamics are extremely brittle. Transfer learning performance on the task of semantic segmentation is lacking.

4.2 Related Work

A detailed review of self-supervised methods was already presented in section 2.1. Here we discuss classical prior work that used motion for segmenting image regions albeit outside the context of self-supervision.

Our method is based on using various motion cues to evaluate image regions. Related to this [61, 21] propose energy based methods to group regions with experiments using temporal geometry cues. [124] develop an algorithm to group motion tracks to represent an object. [146] perform binocular segmentation of independently moving objects. They compute scene flow using optical-flow and disparity, then perform graph cuts on an energy based on motion likelihoods. [34] 's non-local consensus method (NLC) was used as a black-box video segmentation tool in the self-supervision pipeline of [106].

Our S3-CNN approach differs from these methods in its problem definition. We are interested in **learning a model of static images**, using cues such as geometry only to provide supervision. On the contrary, all methods listed above exploit these cues at inference time, and they require motion information at test time. Thus a detailed review of related video segmentation methods is beyond the scope of this thesis.

4.3 Method

We propose a method to pre-train a neural network $\Phi(\mathbf{x})$ operating on individual video frames $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ given corresponding estimates of the optical-flow $\mathbf{f} \in \mathbb{R}^{H \times W \times 2}$.

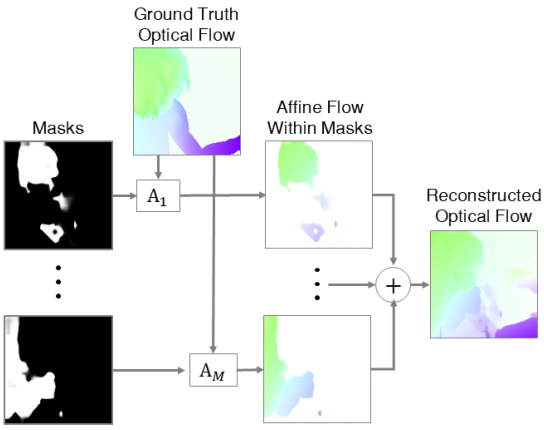


Figure 4.2: Affine optical-flow based self-supervision loss. Optical-flow within each region is approximated using an affine transformation $(A_1, \cdots, A_M)$. These transformations are recombined to give a reconstructed flow which is compared against the ground truth.

Given example pairs $(\mathbf{x}_1, \mathbf{f}_1), \ldots, (\mathbf{x}_N, \mathbf{f}_N)$ of frames and flows, the model Φ is learned by minimizing the energy

$$\min_{\Phi} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\mathbf{f}_i | \Phi(\mathbf{x}_i)) \tag{4.1}$$

where $\mathcal{L}$ is a suitable loss function. If the network $\Phi(\mathbf{x})$ is configured to estimate the posterior density $p(\mathbf{f}|\Phi(\mathbf{x}))$ of the flow $\mathbf{f}$ given image $\mathbf{x}$, then a suitable loss is the negative log-likelihood $\mathcal{L}(\mathbf{f}|\Phi(\mathbf{x})) = -\log p(\mathbf{f}|\Phi(\mathbf{x}))$. Furthermore, if the posterior is unimodal and peaky, it can be modeled as Gaussians $-\log p(\mathbf{f}|\Phi(\mathbf{x})) \propto \|\mathbf{f} - \Phi(\mathbf{x})\|^2$, essentially reducing eq. (4.1) to regression. However, observing a single video frame $\mathbf{x}$ leaves significant uncertainty about the flow $\mathbf{f}$, so that the posterior is complex in practice and the unimodal approximation is poor.

Rather than trying to estimate a complex posterior density directly, our idea is to let the network predict a segmentation $\Phi : \mathbf{x} \mapsto \mathbf{m} \in \{1, \ldots, L\}^{H \times W}$ of the image. $L \in \mathbb{N}$ is the fixed number of segments. This segmentation does not fully determine the posterior, but constrains and simplifies it. In particular, we assume that pixels within each region are *independent and identically distributed (IID)* with respect to a simple parametric distribution. Let $u \in [1, \ldots, H] \times [1, \ldots, W]$ denote pixels and let l index the L segments, then $p(f_u|\theta_l)$ is the simple parametric distribution that determines pixel

flows. Marginalizing the region parameters θ_l results in the model:

$$p(\mathbf{f}|\mathbf{m}) = \prod_{l=1}^{L} \int \left[\prod_{u:m_u=l} p(f_u|\theta_l) \right] p(\theta_l)\, d\theta_l. \tag{4.2}$$

Crucially, due to the marginalization, network Φ is not tasked with predicting the flow parameters θ, but only the regions $\mathbf{m}$.

As a simpler alternative to marginalizing by integration, in the rest of this chapter we marginalize the model parameters by maximization and drop the prior over them. Thus the probability density for a region is written as:

$$p(\mathbf{f}|\mathbf{m}) = \prod_{l=1}^{L} \max_{\theta_l} \prod_{u:m_u=l} p(f_u|\theta_l). \tag{4.3}$$

As a concrete example, if $f_u \in \mathbb{R}$ is the magnitude of flow at pixel u and regions have approximately constant flow magnitude, they can be modeled by Gaussian densities $\log p(f_u|\theta_l) \propto -(f_u - \theta_l)^2$, resulting in the loss:

$$\mathcal{L}(\mathbf{f}|\mathbf{m}) = \sum_{l=1}^{L} \min_{\theta_l} \sum_{u:m_u=l} (f_u - \theta_l)^2 = \sum_{l=1}^{L} \text{variance}\{f_u, : m_u = l\}.$$

Thus, we have replaced the problem of predicting the magnitude of optical-flow with a much simpler one of finding segments with constant flow magnitude.

The above derivation gives us some insights into the nature of our framework. However, in the rest of this chapter we side step probabilities entirely and present loss functions $\mathcal{L}(\mathbf{f}|\mathbf{m})$ that make intuitive sense. Furthermore, to enable backpropagation [116] we use soft-segmentation $\mathbf{m}_l \in [0,1]^{H \times W}\ \forall\ l \in \{1, \ldots, L\}$. In general, our loss functions assess regions by measuring how well simple models can fit them, capturing various forms of region-wide regularity. The parameters of these models are not estimated by the network from image $\mathbf{x}$, but rather during loss evaluation when the optical-flow $\mathbf{f}$ is also available; which is much simpler. Several concrete examples of this idea are given in section 4.3.1 and details about the S3-CNN network are given in section 4.3.2.

4.3.1 Self-Supervised Segmentation Losses

This section introduces three losses $\mathcal{L}(\mathbf{f}|\mathbf{m})$ for self-supervised segmentation. Each loss captures a different type of "coherent" motion: affine, rigid, and low-entropy. All

losses admit a sub-gradient with respect to the mask $\mathbf{m}$ and are therefore suitable for training a deep network using back-propagation [116]. An illustration of the affine motion loss is given in fig. 4.2.

Affine motion loss: For this model, an affine transformation is fitted to the optical-flow within each segment and a robust fitting error is used as loss. In more detail, let (x, y) be the coordinates of a pixel. Each pixel is assigned to a region l in a soft manner, based on its mask weight $\mathbf{m}_l(x, y) \in [0, 1]$. The number of segments $L \in \mathbb{N}$ is a fixed hyper-parameter, $l \in \{1, \ldots, L\}$. Flow vectors of pixels that belong to region l are approximated by a certain region-specific affine transformation[1] $\theta_l \in \mathbb{R}^6$. This yields a reconstructed flow field:

$$\hat{\mathbf{f}}(x, y; \theta_l) = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix} \theta_l$$

The optimal parameters $\{\theta_l^*\}$ are found by minimizing the reconstruction residual:

$$\min_{\theta_1, \ldots, \theta_L} \sum_{l=1}^{M} \mathbf{m}_l(x, y) ||\hat{\mathbf{f}}(x, y; \theta_l) - \mathbf{f}(x, y)||^2.$$

This is a weighted least squares problem, whose solution is given by

$\theta_l^* = (X^\top M_l X)^{-1} X^T M_l F$ where

$$X = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{HW} & y_{HW} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{HW} & y_{HW} & 1 \end{bmatrix}, \quad F = \begin{bmatrix} \mathbf{f}(x_1, y_1) \\ \vdots \\ \mathbf{f}(x_{HW}, y_{HW}) \end{bmatrix},$$

and $M_l = \mathrm{diag}(\mathbf{m}_l(x_1, y_1), \mathbf{m}_l(x_1, y_1), \ldots, \mathbf{m}_l(x_{HW}, y_{HW}), \mathbf{m}_l(x_{HW}, y_{HW}))$ is the diagonal matrix of mask values.

To define our loss function, we compute the net reconstructed flow by combining per segment affine flows (Figure 4.2):

$$\hat{\mathbf{f}}(x, y; \{\theta_l^*\}) = \sum_{l=1}^{L} \mathbf{m}_l(x, y) \hat{\mathbf{f}}(x, y; \theta_l^*)$$

[1] Normally for an affine transformation A, $(x_2, y_2) = A[x_1; y_1; 1]$ but, since the flow is given by $(f_x, f_y) = (x_2 - x_1, y_2 - y_1)$, we have $\theta_l = \text{reshape}\left(A - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\right)$

For added stability, the loss is then a robust function of the residual between this reconstruction and the ground truth.

$$\mathcal{L}(\mathbf{f}|\mathbf{m}) = R(\hat{\mathbf{f}} - \mathbf{f}) \tag{4.4}$$

where R is the smooth-L1 function [45]. Intuitively, if the projected motion inside each region is affine, then the reconstructed flow will match the ground truth and the loss will be 0. As a special case, this occurs if all predicted regions contain planar objects under orthographic projection.

Computing eq. (4.4) involves basic arithmetic and algebraic operations (such as matrix-vector multiplication and matrix inverse). All of these operations are differentiable [109] and implemented in TensorFlow [1].

Rigid motion: Our second loss function is suitable for arbitrary 3D motions of rigid objects imaged by a perspective camera. We require that the motion field within each segment satisfy an **epipolar constraint**. To this end, we use the eight-point algorithm [86, 54] to estimate the fundamental matrix $\mathcal{F}_l$ for each region using dense correspondences induced by the optical-flow vectors. The resulting algebraic residual is our loss. Detailed derivation follows.

Let $q := (x, y)$ be pixel coordinates of a rigid object moving with optical-flow $\binom{f_x}{f_y} = \mathbf{f}(x, y) \in \mathbb{R}^2$. This induces correspondences:

$$q \to (x + f_x, y + f_y)$$

Define $q' := (x', y') := (x + f_x, y + f_y)$. This satisfies the epipolar constraint:

$$\begin{bmatrix} q'^T 1 \end{bmatrix} \mathcal{F} \begin{bmatrix} q \\ 1 \end{bmatrix} = 0 \tag{4.5}$$

where $\mathcal{F} \in \mathbb{R}^{3\times3}$ is the fundamental matrix. The 8-point algorithm [86, 54], reformulates this as a minimum norm problem. We review its derivation first before discussing our loss function.

Let:

$$\mathcal{F} := \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{23} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

and $\beta := \begin{bmatrix} f_{11} \\ \vdots \\ f_{33} \end{bmatrix} \in \mathbb{R}^9$ is the serialized version of $\mathcal{F}$. Using these, the epipolar constraint can be rewritten, jointly for all correspondences, as $A\beta = 0$ where

$$A = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_Nx_N & x'_Ny_N & x'_N & y'_Nx_N & y'_Ny_N & y'_N & x_N & y_N & 1 \end{bmatrix}$$

The 8-point algorithm computes β^* as the minimizer:

$$\beta^* = \min_{\beta} ||A\beta||_2^2 \text{ such that } ||\beta||_2 = 1 \tag{4.6}$$

Note that, $||A\beta||_2^2 = \beta^T A^T A\beta$. Solving eq. (4.6) by using the Rayleigh-Quotient theorem gives, β^* as the eigen vector corresponding to the smallest eigen value of A^TA. Equivalently, β^* is the singular vector corresponding to the smallest singular value of A.

We adapt this derivation for our loss to use a per-pixel soft-mask $\mathbf{m}(x, y) \in [0, 1]$ by estimating β as

$$\beta^* = \min_{\beta} ||M^{1/2}A\beta||_2^2 \text{ such that } ||\beta||_2 = 1 \tag{4.7}$$

where $M = \text{diag}(\mathbf{m}(x_1, y_1), \ldots, \mathbf{m}(x_{HW}, y_{HW}))$ is the diagonal matrix of mask values. Similar to the 8-point algorithm, $||M^{1/2}A\beta||_2^2 = \beta^T A^T MA\beta$. Since M is a diagonal matrix, A^TMA is symmetric and the Rayleigh-Quotient theorem can be applied. Therefore, β^* is the eigen vector corresponding to the smallest eigen value of A^TMA.

Our loss function is based on the algebraic residual with respect to β^*. Let $\mathcal{F}(\beta^*) \in \mathbb{R}^{3\times3}$ be the fundamental matrix corresponding to parameters in β^*. Then,

$$\mathcal{L}(\mathbf{f}|\mathbf{m}) = R\left(\begin{bmatrix} q'^T 1 \end{bmatrix} \mathcal{F}(\beta^*) \begin{bmatrix} q \\ 1 \end{bmatrix}\right) \tag{4.8}$$

where R is the smooth-L1 function [45].

Epipolar geometry also requires that the determinant of $\mathcal{F}$ be zero, $\det(\mathcal{F}) = 0$. This is typically imposed using a singular value decomposition (SVD), by setting the smallest singular value to 0. We did not impose this constraint to avoid the complexity of back-propagating through the SVD. Furthermore, the 8-point algorithm normalizes correspondences before computing A and its singular value decomposition. This is required for numerical stability. We found that normalization hinders convergence when

training the S3-CNN and do not normalize correspondences. We use double precision arithmetic for computing $Q := A^T M A$ and its eigen decomposition. Numerical error can lead to Q being non-symmetric. We compensate for this by symmetrizing it after the fact as $Q \leftarrow \left(Q + Q^T\right) / 2$. Backpropagation through this loss function requires differentiating an eigen-decomposition, which is also implemented in TensorFlow.

Low entropy motion: Instead of fitting parametric models to regions, histograms offer a general non-parametric alternative. To use them, we discretize optical-flow into B bins; $f_u \in \{1, \ldots, B\}$. Our loss function computes the *weighted entropy of the empirical flow distribution over regions*. We first write the equations for hard assignment segmentation $\mathbf{m} \in \{1, \ldots L\}^{H \times W}$:

$$\mathcal{L}(\mathbf{f}|\mathbf{m}) = \sum_{l=1}^{L} w_l H(\hat{p}_l), \quad w_l = \sum_{u:m_u=l} 1 \tag{4.9}$$

where $H(\hat{p}_l) = -\sum_{b=1}^{B} \hat{p}_l(b) \log \hat{p}_l(b)$ is the entropy of the histogram obtained by normalizing the flow counts for region l.

$$\hat{p}_l(b) \propto \sum_{u:m_u=l} \delta(f_u = b) \; \forall \; b \in \{1, \ldots, B\}$$

This easily extends to the case of soft mask weights $\mathbf{m}_u^l \in [0, 1]$ using, $w_l = \sum_u \mathbf{m}_u^l$ and $\hat{p}_l(b) \propto \sum_u \mathbf{m}_u^l \, \delta(f_u = b)$. In our experiments, we use this model with flow magnitudes $|f_u| \in \mathbb{R}$ instead of 2D optical-flow vectors. In order for the quantization of flow magnitude to capture both subtle and large motions, we normalize it logarithmically before binning, using

$$\tilde{f}_u = \min(1, \log(|f_u| + 1) / \log(M + 1)) \tag{4.10}$$

where $M = 50$ pixels is a cut-off value. The loss function then becomes,

$$\mathcal{L}(\mathbf{f}|\mathbf{m}) = \sum_{l=1}^{L} w_l H(\hat{p}_l), \quad w_l = \sum_{u \in \Omega} \mathbf{m}_u^l, \quad \hat{p}_l(b) \propto \sum_u \mathbf{m}_u^l \, \delta(\tilde{f}_u = b) \tag{4.11}$$

The above derivations may confuse the reader about the method itself. It is very simple in practice as summarized below.

Summary

1. Compute $\mathbf{m} = \phi(\mathbf{x})$

2. Estimate θ_l parameters of the motion model as an affine-transform / fundamental matrix / histogram using inputs $\mathbf{m}$ and $\mathbf{f}$.

3. Compute loss using either of equations (4.4), (4.8), (4.11).

4.3.2 CNN Architecture for Segmentation

There are a variety of CNN architectures suitable for image segmentation, and thus for implementing the learn-able function $\Phi(\mathbf{x})$ in eq. (4.1). We opt for a relatively simple Fully Convolutional Network (FCN) [85], and specifically the FCN-8s model on VGG-16 [122]. The FCN architecture is well suited for our experiments because it does not change the core classification network except for making the fully connected layers into convolutions. This allows us to directly compare our features against existing self-supervised learning methods and the corresponding ImageNet ILSVRC [117] trained model.

FCN scores are mapped to soft segmentation masks as in [37]. We first apply a non-linearity to map values into $[-1, 1]$. Given pixel scores s_u, these are computed as

$$m_u = 2\sigma(s_u) - 1$$

where σ is the sigmoid function. This is followed by a per-pixel softmax with temperature β. We use $\beta = 30$ or $\beta = 10$ in all our experiments. This non-conventional segmentation head is necessary to make the model converge even on toy data, possibly indicating badly behaved gradients in such network architectures.

4.4 Experiments

We demonstrate qualitative results of our framework on learning image segmentation using optical-flow cues in section 4.4.2. We assess its capability to pre-train for image recognition in section 4.4.3.

4.4.1 Dataset Details

Single-cube: This synthetic toy dataset consists of a single translating and rotating 3D textured cube (fig. 4.3a); paired with the corresponding optical-flow field. Cubes are imaged under an orthographic camera. This is a small dataset of just five sequences with 99 frames each. It is so small that all of the frames had to be used for training. Example images can be seen in fig. 4.3a. This dataset was rendered using Panda3D [48]. Ground truth optical-flow is computed using the code of [130].

Multi-cube: This is a synthetic toy dataset consisting of multiple cubes under perspective projection. The cubes are rigid. We generated 40 sequences of 198 frames each for training and held out 5 sequences of 198 frames each for testing. Example frames can be seen in fig. 4.3b. This dataset was rendered using Panda3D [48]. Ground truth optical-flow is computed using the code of [130].

Sintel: The Sintel dataset [11] is based on a blender open movie `https://durian.blender.org/`. It contains 23 sequences of which we use 20 for training[2] and 3 for testing[3], and has ground-truth optical-flow computed by Blender™ itself. This dataset is synthetic as well although it is much more diverse than the moving cube data above. Objects in the scene are typically non-rigid, including people, dragons and background. This dataset consists of two sets of images - "clean" and "final". The latter contains effects such as smoke and motion blur to make it look realistic. The former does not. We used Sintel "clean" in our experiments, but we do not expect a significant change when switching to Sintel "final".

YFCC100m-150k: The YFCC100m [131] dataset consists of around 800k real-world, in-the-wild videos. We derive a dataset of optical-flow-RGB frame pairs from it. To this end, we sample the first and fifth frame from around 150k videos and compute

[2]Training sequences: alley_1, alley_2, ambush_2, ambush_4, ambush_5, ambush_6, ambush_7, bamboo_1, bamboo_2, bandage_1, bandage_2, cave_2, cave_4, market_2, market_5, market_6, mountain_1, shaman_2, shaman_3, sleeping_1

[3]Testing sequences: sleeping_2, temple_2, temple_3

optical-flow between them using the EpicFlow algorithm [114][4], with initial matches obtained using FlowFields [5]. This dataset is used for training only.

Youtube objects: The Youtube Objects dataset [111] consists of Youtube videos. We extract 5k frames and compute optical-flow using [114, 115]. This is the same as the dataset in section 3.3.2 and is used as a validation set only.

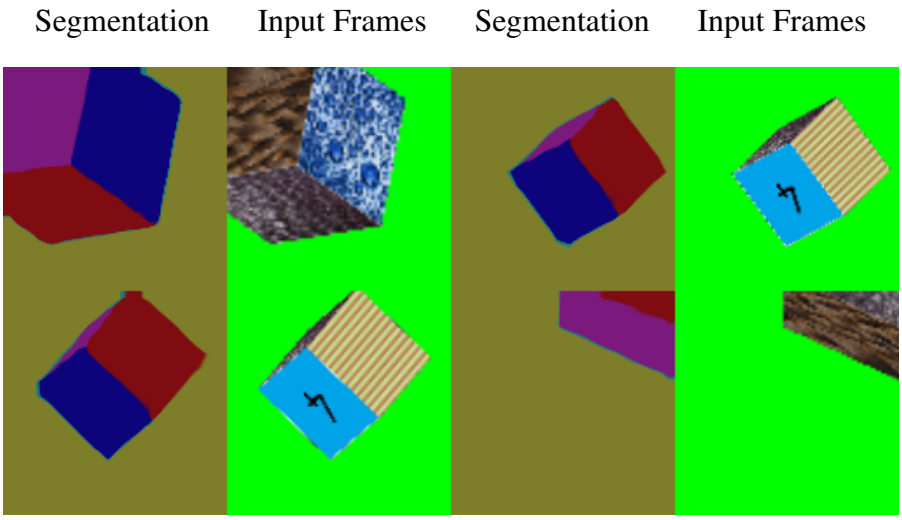
Data augmentation: We pretrain VGG-16 FCN-8s with crops of size 224×224. These crops are obtained by first randomly resizing the image to 1.01x - 1.5x the crop-size. This is followed by a random crop and a random left-right flip. For the YFCC100m-150k dataset, we also do colour augmentation: shifting the hue by up to 0.1, random contrast between 0.2 and 1.8, random brightness by up to 0.12 (based on $0-1$ normalised colours).

4.4.2 Qualitative Results

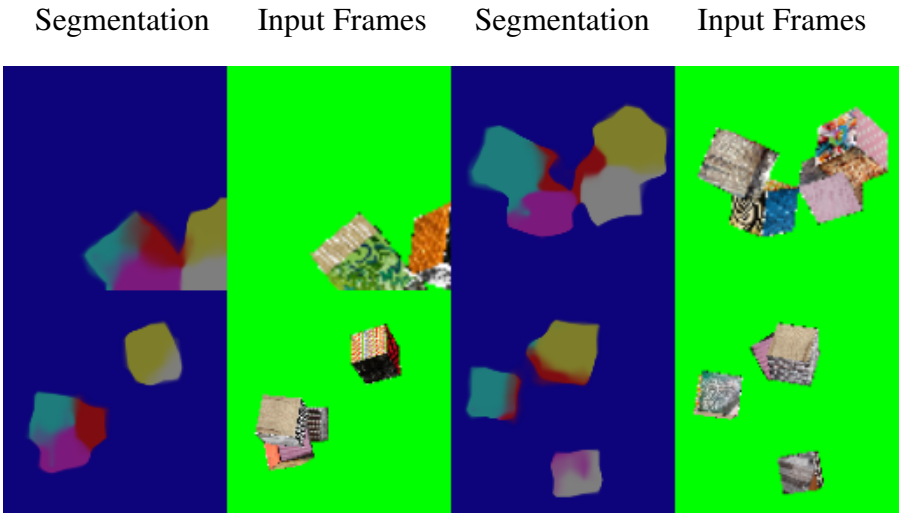
Moving cubes: First, we demonstrate our method on the toy single-cube and multi-cube datasets. In the single-cube case, projections of individual cube faces are undergoing affine transformations as they are imaged under orthographic projection. We therefore apply the affine motion model of section 4.3.1 and learn a model, without manual supervision, that segments individual cube faces as shown in fig. 4.3a. In the multi-cube case, the epipolar constraint discussed in section 4.3.1 is applicable. Our network is able to learn to segment individual independently moving cubes in many cases even under severe occlusion (fig. 4.3b).

Sintel: The Sintel dataset is much more challenging. An affine approximation is poor for the motion of non-rigid objects under perspective projection. We still apply the affine motion model to it because when given a limited number of segments it is forced to group together complex motion patters. We observe that these groupings turn out to be very reasonable objects/object-parts. The results obtained are shown in fig. 4.4a, where several such objects and parts are highlighted. Notice in particular that even bodies and heads are picked up despite their complex motion.

[4]Epic flow uses structured edge detection to obtain an edge map. This is trained using manually annotated data. We assume that the influence of this supervision is weak and does not affect our main results



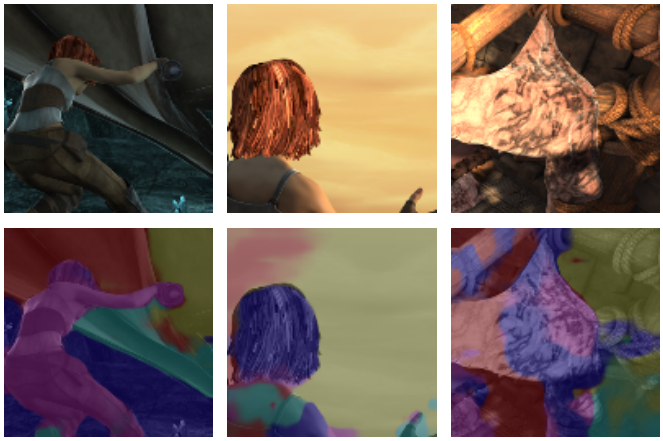
(a) Orthographic projection: Segmenting cube faces (train set).



(b) Perspective projection: Segmenting entire cubes (validation set)

Figure 4.3: Moving cubes: These 2×4 grids of images show example input frames and their predicted segmentation masks. The segmentation masks are visualized as a colour map to the left of each input frame. Two rows of examples are shown for each dataset, 4 input frames per dataset in total. View in colour.

YFCC100m-150k: In the case of real world data, there is large systematic noise in automatically computed optical-flow. We find that the histogram entropy loss applied on flow magnitudes works best in these cases. The affine and rigid models failed in preliminary experiments but could be reconsidered when unsupervised optical-flow methods improve. We trained our flow magnitude histogram entropy model on the YFCC100m-150k dataset and visualized predicted segments on the Youtube Objects dataset in fig. 4.4b. The cat boundaries align well with segments in the first column. Note



Sintel - 5 regions

Youtube Objects - 10 regions - Trained on YFCC100m-150k

Figure 4.4: S3-CNN output for different datasets on validation sets, except first column for Sintel. For each dataset, the top row shows input frames and the bottom row overlays the segmentation masks, visualized as colour maps, over the input frames. View in colour.

a bird in the middle is segmented out and so is a cat in the second column suggesting that the model has a liking for object like regions. Also each segment caters to one spatial region. The teal coloured region is always on the middle left whereas the light green region is always on the top right corner. This shows that the model has learned to be non-convolutional despite a fully-convolutional architecture. We believe that this is because of the instance specific nature of our proxy-task where individual instances of the same category may move independently. In-fact motion patterns may not contain instance agnostic semantic information. Another reason is that the model is unaware of which segments are free and which are occupied by other objects in the scene. Given two independently movable objects it needs to arbitrarily decide a mapping from objects to segment numbers. The loss itself is invariant to this assignment.

4.4.3 Self-Supervision for Object Recognition

The *S3* framework can also be used as a proxy to pre-train a generic feature extractor. We pre-train a VGG-16 FCN-8s model on the YFCC100m-150k dataset using the flow magnitude histogram entropy minimization loss. The Youtube objects dataset is used to monitor validation error to control over-fitting. Features from this pre-trained network are then fine-tuned for recognition tasks: Pascal VOC 2007 classification and VOC 2012 segmentation. These self-supervision benchmarks are identical to those in section 3.3.5.

The VOC-2007 classification task is a multi-class image classification problem with 20 classes. Mean average precision is reported on the Pascal VOC 2007 test set. We follow the protocol of [74] except that we double the number of iterations to 160k and therefore double the learning rate steps as well - $20k, 40k, 60k, 80k, 100k, 120k, 140k$.

The VOC-2012 semantic segmentation task requires performing pixel-level segmentation of the 20 VOC classes and a background class. For fair comparison with [78], we use their sparse hypercolumn model [77, 6] with large field of view (FoV) by adding two sets of conv-pool modules at the end of our VGG-16 pre-trained network. Note that this large FoV trick was not used in our AlexNet experiments of chapter 3. We use the standard augmented 2012 segmentation data [52] and training-validation splits with 10582 training images and 1449 validation images respectively. We train on the training split and report mean intersection over union (mIoU) on the validation split. The learning rate schedule is adapted as in [78], detailed in section 3.3.5. Parameter free batch normalization [60] is used after every convolutional and fully connected layer in the pre-training stage. Batch normalization moments are absorbed into convolution filters and biases before fine-tuning.

Table 4.1 compares against other methods that report results on VOC-07 classification and VOC-12 segmentation using a VGG-16 based model. We observe that our S3-CNN-YFCC100m-150k[6] model performs better than a non pretrained VGG-16. We are performing competitive to state-of-the-art models for VOC-2007 classification. We get

[6]S3-CNN-YFCC100m-150k was initially trained on lower resolution images (128x128) without batch normalisation in the fully connected layers (fc6-7). We trained for 4M iterations on the YFCC100m-150k dataset using ADAM [73] with initial learning rate of 10^{-4}, with a drop to 10^{-5} at 3.25M iterations. We then re-initialized the fully connected layers (fc6-7) and added batch normalization to them. This model was further trained on higher resolution (224x224) YFCC100m-150k images for 640k iterations using ADAM with initial learning rate 10^{-5}.

Method	Architecture	VOC07 %mAP	VOC12 %mIoU
ImageNet	VGG-16 (+FoV)	86.9	69.5
Random ($\sim$ [78][5])	VGG-16 (+FoV)	59.9	33.6
k-means [74]	VGG-16	56.5	-
Colorization [77]	VGG-16	-	50.2
Surface Normals [6]	VGG-16	-	52.4
Colorization [78]	AlexNet	65.9	38.4
Colorization [78]	VGG-16 (+FoV)	77.2	56.0
S3-CNN-YFCC100m-150k	VGG-16 (+FoV)	76.4	45.5

Table 4.1: VOC Comparison: We fine-tune our model for VOC2007 classification (results on test split) and VOC2012 Segmentation (results on validation split). (+FoV) means that two extra conv-pool layers were added after the VGG-16 fc7 layer before computing the hyper column for segmentation. It does not influence the classification results.

76.35% mAP compared to 77.2%mAP of [78] despite using only 150k pre-training pairs compared to their pre-training dataset of 3.7M images. We lag behind on VOC-2012 segmentation. This might possibly be because, in our case, the pre-trained model is a non-semantic instance segmentation network whereas the finetuned model needs to be an instance agnostic semantic segmentation network. Furthermore our model uses image boundaries as an anchor for assigning regions to segments. The top left region, for instance, is always assigned to segment 1, say. Unlearning these traits and retaining visual concepts might be challenging for the network and might explain the weaker semantic segmentation results. Lastly, we trained an AlexNet model akin to that of [106] by constructing an AlexNet FCN32s S3-CNN. We compare with [106] on VOC-07 classification and obtain 57.37% mAP versus their reported result of 61% mAP. While we directly use optical-flow, Pathak *et al.* [106] use the heavy machinery of uNLC [34] on top of it and pre-train using 1.6M images. Our AlexNet architectures are identical to theirs except that when finetuning we retain the 100 pixel padding in conv-1 and do a global average pool after fc7. This is to comply with the boundary sensitivity of our pre-trained network. Note that 57.37% is significantly less than 65.3% obtained by the CPFS-KTA model of chapter 3. This is partially due to the difference in pre-training dataset sizes. Unfortunately S3-CNNs take several months to train and we could not

perform a large scale experiment. The poor performance may also be attributed to poor convergence properties of S3-CNN training and the nature of our flow consistency loss.

4.5 Discussion

We have presented the *S3* framework, that allows supervising neural network architectures using more sophisticated consistency measures such as affine and rigid motion. We observe that this worked well when optical-flow is perfect as in synthetic data. We adapted our model to use flow magnitude histograms which worked for real-world computed optical-flow. Transfer learning for recognition tasks shows promising results for VOC-2007 classification but poor mean intersection over union scores for VOC-2012 semantic image segmentation. We observe that S3-CNN training is slow and therefore useable only in specific small scale applications where certain transformation classes are relevant.

4.6 Statement of Authorship

This chapter is based on joint work with James Thewlis (CDT, Department of Engineering Science, University of Oxford). Although all work was done collaboratively as a joint effort, his individual contributions are detailed below:

1. Prepared and computed the YFCC100m-150k dataset (See section 4.4.1).
2. Formulated and implemented the low entropy flow magnitude histogram loss (See section 4.3.1).
3. Pre-trained S3-CNN on YFCC100m-150k dataset.
4. Designed and rendered dataset for moving cubes using panda 3D (See section 4.4.1).
5. Developed and engineered the flow normalization method (eq. (4.10)).

5 Visualizing Deep CNNs Using Natural Pre-Images

In this thesis, we have so far presented two methods for self-supervised learning using motion cues. The first of these, CPFS-CE, performed well when transferring the pre-trained model for recognition tasks. We, however, lack an intuitive understanding of the neurons themselves prior to the transfer. This is particularly concerning given that the ILSVRC-12 linear probing experiments (table 3.2), which freeze the underlying representation, failed to yield competitive results. In this chapter we take a detour to develop a suite of visualization techniques which will be used to visualize the CPFS-CE model in chapter 6.

5.1 Introduction

Most image understanding and computer vision methods do not operate directly on images, but on suitable image representations. Notable examples of representations include textons [82], histogram of oriented gradients (SIFT [88] and HOG [17]), bag of visual words [16, 125], sparse [153] and local coding [143], super vector coding [165], VLAD [66], Fisher Vectors [108], and, lately, deep neural networks, particularly of the convolutional variety [76, 156, 119]. While the performance of representations has been improving in the past few years, their design remains eminently empirical. This is true for

shallower hand-crafted features such as HOG and SIFT and even more so for the latest generation of deep representations, such as deep Convolutional Neural Networks (CNNs), where millions of parameters are learned from data. A consequence of this complexity is that our understanding of such representations is limited. In this chapter, with the aim of obtaining a better understanding of representations, we develop a family of methods to investigate CNNs and other image features by means of visualizations. All these methods are based on the common idea of seeking natural-looking images whose representations are notable in some useful sense. We call these constructions *natural pre-images* and propose a unified formulation and algorithm to compute them (section 5.2).

The suite of visualization methods explored in this chapter are the following:

Inversion: This method computes the "inverse" of a representation. This is done by modeling the representation as a function $\Phi_0 = \Phi(\mathbf{x}_0)$ of the image $\mathbf{x}_0$, and attempting to recover the image from the information contained only in the code Φ_0.

Activation maximization: This method optimizes a randomly initialized pre-image $\mathbf{x}$ so that it maximizes the response of a certain neuron $[\Phi(\mathbf{x})]_i$ in the representation. The resulting image visualizes the input stimuli that strongly activate a neuron. It helps understand the neuron's "meaning" or function.

Caricaturization: Here the initial image $\mathbf{x}_0$ is updated to exaggerate any visual patterns that positively activate neurons in the representation $\Phi(\mathbf{x}_0)$. Differently from activation maximization, this visualization method emphasizes the meaning of combinations of neurons that are active together.

In the rest of this chapter, neuron, component and unit are used interchangeably to mean the same thing. Also, positive activation, activation and excitation of a neuron all mean that the neuron's output is positive. In this context negative activations are referred to as 'inhibition'.

Our first contribution is the idea of a *natural pre-image*, i.e. to restrict pre-images to the set of natural images. While this is difficult to achieve in practice, we explore

different regularization methods (section 5.2.2) that can work as a proxy, including regularizers using the Total Variation (TV) norm of the image. We also explore an indirect regularization method, namely the application of random jitter to the reconstruction as suggested by Mordvintsev *et al.* [93].

Our second contribution is the inversion of neural networks using these regularization based natural image priors. We are the first to visualize modern deep feedforward CNNs using such priors (section 5.4.2).

Our third contribution is to consolidate different visualization types, including inversion, activation maximization, and caricaturization, in a common framework (section 5.2). We propose a single algorithm applicable to a large variety of representations, from SIFT to very deep CNNs, using essentially a single set of parameters. The algorithm is based on optimizing an energy function using gradient descent and back-propagation through the representation architecture.

Our fourth contribution is to apply these three visualization types to the study of several different representations. First, we show that, despite its simplicity and generality, our method recovers significantly better reconstructions for shallow representations such as HOG compared to recent alternatives [137] (section 5.4.1). In order to do so, we also rebuild the HOG and DSIFT representations as equivalent CNNs, simplifying the computation of their derivatives as required by our algorithm (section 5.3.1). Second, we apply inversion (section 5.4.2), activation maximization (section 5.5), and caricaturization (section 5.6) to the study of CNNs, treating each layer of a CNN as a different representation, and studying different state-of-the-art architectures, namely AlexNet, VGG-M, and VGG Very Deep (VGG-VD-16 section 5.3.2). As we do so, we emphasize a number of general properties of such representations, as well as differences between them. In particular, we study the effect of depth on representations, showing that CNNs gradually build increasing levels of invariance and complexity, layer after layer.

Our findings are summarized in section 5.7. The code for the experiments in this chapter and extended visualizations are available at `http://www.robots.ox.ac.uk/~vgg/research/invrep/index.html`. This code uses the open-source

MatConvNet toolbox [136] and publicly available copies of the models to allow for easy reproduction of results.

5.2 Pre-Image Method

This section introduces our method to find pre-images of an image representation. This method will then be applied to the inversion, activation maximization, and caricaturization problems. Our method is a regularized energy minimization where the goal is to find a natural-looking image whose representation has a desired property [150]. Formally, given a representation function $\Phi : \mathbb{R}^{H\times W\times D} \to \mathbb{R}^d$ and a reference code $\Phi_0 \in \mathbb{R}^d$, we seek the image[1] $\mathbf{x} \in \mathbb{R}^{H\times W\times D}$ that minimizes the objective function:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^{H\times W\times D}} \mathcal{R}_\alpha(\mathbf{x}) + \mathcal{R}_{TV^\beta}(\mathbf{x}) + C\,\ell(\Phi(\mathbf{x}), \Phi_0) \tag{5.1}$$

The loss ℓ compares the image representation $\Phi(\mathbf{x})$ with the target value Φ_0, the two regularizer terms $\mathcal{R}_\alpha + \mathcal{R}_{TV^\beta} : \mathbb{R}^{H\times W\times D} \to \mathbb{R}_+$ capture a *natural image prior*, and the constant C trades off loss and regularizers.

The meaning of minimizing the objective function eq. (5.1) depends on the choice of the loss and of the regularizer terms, as discussed below. While these terms contain several parameters, they are designed such that, in practice, all the parameters except C can be automatically set for all visualization and representation types.

5.2.1 Loss Functions

Choosing different loss functions ℓ in eq. (5.7) results in different visualizations. In *inversion*, ℓ is set to the Euclidean distance:

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \frac{\|\Phi(\mathbf{x}) - \Phi_0\|^2}{\|\Phi_0\|^2}, \tag{5.2}$$

where $\Phi_0 = \Phi(\mathbf{x}_0)$ is the representation of a probe image. Minimizing eq. (5.1) results in an image $\mathbf{x}^*$ that "resembles" $\mathbf{x}_0$ from the viewpoint of the representation.

Sometimes it is interesting to restrict the loss to a subset of the representation components. This could be a subset of the feature channels or a subset of spatial locations.

[1] In the following, the image $\mathbf{x}$ is assumed to have null mean, as required by most CNN implementations.

Figure 5.1: Input images used in the rest of this chapter are shown above. From left to right: spoonbill, gong, monkey, building, red fox, abstract art.

This restriction is done by introducing a binary *mask* M of the same dimension as Φ_0 and by modifying eq. (5.2) as follows:

$$\ell(\Phi(\mathbf{x}), \Phi_0; M) = \frac{\|(\Phi(\mathbf{x}) - \Phi_0) \odot M\|^2}{\|\Phi_0 \odot M\|^2}, \tag{5.3}$$

In *activation maximization* and *caricaturization*, $\Phi_0 \in \mathbb{R}^d_+$ is treated instead as a weight vector selecting which representation components should be maximally activated. This is obtained by considering the inner product:

$$\ell(\Phi(\mathbf{x}), \Phi_0) = -\frac{1}{Z}\langle \Phi(\mathbf{x}), \Phi_0 \rangle. \tag{5.4}$$

For example, if $\Phi_0 = \mathbf{e}_i$ is the indicator vector of the i-th component of the representation, minimizing eq. (5.4) maximizes the component $[\Phi(\mathbf{x})]_i$. This is called activation maximization. Alternatively, if Φ_0 is set to $\max\{\Phi(\mathbf{x}_0), 0\}$, the minimization of eq. (5.4) will highlight components that are active in the representation $\Phi(\mathbf{x}_0)$ of a reference image $\mathbf{x}_0$, while ignoring the inactive components. This is called caricaturization.

The choice of the normalization constant Z in activation maximization and caricaturization will be discussed later. Note also that, for the loss eq. (5.4), there is no need to define a separate mask as this can be pre-multiplied into Φ_0.

5.2.2 Regularization

Discriminative representations discard a significant amount of low-level image information that is irrelevant to the target task (e.g. image classification). As this information is nonetheless useful for visualization, we propose to partially recover it by restricting the inversion to the subset of natural images $\mathcal{X} \subset \mathbb{R}^{H \times W \times D}$. This is motivated by the fact that, since representations are applied to natural images, there is comparatively little

Figure 5.2: *Left:* Effect of the data term strength C in inverting a deep representation (the relu3 layer in AlexNet). Selecting a small value of C results in more regularized reconstructions. This is essential to obtain more interpretable results. *Right:* Effect of the TV regularizer β exponent; note the spikes for $\beta = 1$. The input image in this case is the "spoonbill" image shown in fig. 5.1.

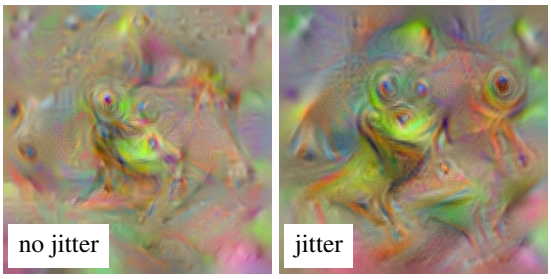


Figure 5.3: Effect of the jitter regularizer in activation maximization for the "tree frog" neuron in the fc8 layer in AlexNet. Jitter helps recover larger and crisper image structures.

interest in understanding their behavior outside of this set. However, modeling the set of natural images is a significant challenge in its own right. As a proxy, we propose to regularize the reconstruction by using simple *image priors* implemented as regularizers in eq. (5.1). We experiment in particular with three such regularizers, discussed next.

Bounded range: The first regularizer encourages the intensity of pixels to stay bounded. This is important for networks that include normalization layers, as in this case arbitrarily rescaling the image range has no effect on the network output. In activation maximization, it is even more important for networks that do *not* include normalization layers, as in this case increasing the image range may increase neural activations by the same amount.

For color images we make the term isotropic in RGB space by considering the norm

$$N_\alpha(\mathbf{x}) = \frac{1}{HWB^\alpha} \sum_{v=1}^{H} \sum_{u=1}^{W} \left(\sum_{k=1}^{D} \mathbf{x}(v,u,k)^2 \right)^{\frac{\alpha}{2}} \tag{5.5}$$

where v indexes the image rows, u the image columns, and k the color channels. The term is normalized by the image area HW and by the scalar B. This scalar is set to the

typical L^2 norm of the pixel RGB vector, such that $N_\alpha(\mathbf{x}) \approx 1$. The soft constraint $N_\alpha(\mathbf{x})$ is combined with a hard constraint to limit the pixel intensity to be at most B_+:

$$R_\alpha(\mathbf{x}) = \begin{cases} N_\alpha(\mathbf{x}), & \forall v,u : \sqrt{\sum_k \mathbf{x}(v,u,k)^2} \leqslant B_+ \\ +\infty, & \text{otherwise.} \end{cases} \tag{5.6}$$

While the hard constraint may seem sufficient, in practice it was observed that without soft constraints, pixels tend to saturate in the reconstructions.

Bounded variation: The second regularizer is the *total variation* (TV) $\mathcal{R}_{TV^\beta}(\mathbf{x})$ of the image, encouraging reconstructions to consist of piece-wise constant patches. For a discrete image $\mathbf{x}$, the TV norm is approximated using finite differences as follows:

$$\mathcal{R}_{TV^\beta}(\mathbf{x}) = \frac{1}{HWV^\beta} \sum_{uvk} \Big((\mathbf{x}(v,u+1,k) - \mathbf{x}(v,u,k))^2 + (\mathbf{x}(v+1,u,k) - \mathbf{x}(v,u,k)))^2 \Big)^{\frac{\beta}{2}}$$

where $\beta = 1$. Here the constant V in the normalization coefficient is the typical value of the norm of the gradient in the image.

The standard TV regularizer, obtained for $\beta = 1$, was observed to introduce unwanted "spikes" in the reconstruction, as illustrated in fig. 5.2 (right), when inverting a layer of a CNN. This is a known problem in TV-based image interpolation (see *e.g.* Figure 3 in [14]). In image interpolation, "spikes" occur at the locations of the samples because: (1) the TV norm along any path between two samples depends only on the overall amount of intensity change (not on the rate of change) and (2) integrated on the 2D image, it is optimal to concentrate changes around a boundary with a small perimeter. Hyper-Laplacian priors with $\beta < 1$ are often used as a better match of the gradient statistics of natural images [75], but they only exacerbate this issue. Instead, we trade off the sharpness of the image with the removal of such artifacts by choosing $\beta > 1$ which, by penalizing large gradients, distributes changes across regions rather than concentrating them at a point or a curve. We refer to this as the TV^β regularizer. As seen in fig. 5.2 (right), the spikes are removed for $\beta = 1.5, 2$ but the image is blurrier than for $\beta = 1$. At the same time, fig. 5.2 (left) illustrates the importance of using the TV^β regularizer in obtaining clean reconstructions. For $\beta = 2$ this is equal to the quadratic image prior.

Jitter: The last regularizer, which is adapted from [93], has an implicit form and consists of randomly shifting the input image before feeding it to the representation. In more detail, we consider the optimization problem

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^{H\times W\times D}} \mathcal{R}_\alpha(\mathbf{x}) + \mathcal{R}_{TV^\beta}(\mathbf{x}) + C\,\mathbb{E}_\tau[\ell(\Phi(\text{jitter}(\mathbf{x};\tau)),\Phi_0)] \tag{5.7}$$

where $\mathbb{E}[\cdot]$ denotes expectation and $\tau = (\tau_1, \tau_2)$ is a discrete random variable uniformly distributed in the set $\{0,\dots,T-1\}^2$, expressing a random horizontal and vertical translation of at most $T-1$ pixels. The $\text{jitter}(\cdot)$ operator translates and crops $\mathbf{x}$ as follows:

$$[\text{jitter}(\mathbf{x};\tau)](v,u) = \mathbf{x}(v+\tau_2, u+\tau_1)$$

where $1 \leqslant v \leqslant H - T + 1$ and $1 \leqslant u \leqslant W - T + 1$. The expectation over τ is not computed explicitly; instead each iteration of SGD samples a new value of τ. Jittering counterbalances the very significant down-sampling performed by the earlier layers of deep CNNs, interpolating between pixels in back-propagation. This generally results in pre-images with more content and less noise, particularly in the activation maximization problem (Example in fig. 5.3).

Texture and style regularizers: For completeness, we note that eq. (5.1) can also implement the texture synthesis and style transfer visualizations of [41, 42]. Their texture/style term can be incorporated into our formulation as an additional regularizer of the form

$$\mathcal{R}_{\text{tex}}(\mathbf{x}) = \sum_{l=1}^{L} w_l ||\psi\circ\Phi_l(\mathbf{x}) - \psi\circ\Phi_l(\mathbf{x}_{\text{tex}}))||^2_{\text{fro}} \tag{5.8}$$

where $\mathbf{x}_{\text{tex}}$ is a reference image defining a texture or "visual style", $\Psi_l, l = 1,\dots,L$ are increasingly deep layers in a CNN, $w_l \geqslant 0$ weights, and ψ is the *cross-channel correlation* operator

$$[\psi\circ\Phi_l(\mathbf{x})]_{cc'} = \sum_{uv} [\Phi_l(\mathbf{x})]_{uvc}[\Phi_l(\mathbf{x})]_{uvc'}$$

where $[\Phi_l(\mathbf{x})]_{uvc}$ denotes the c-th feature channel activation at location (u,v).

The term $\mathcal{R}_{\text{tex}}(\mathbf{x})$ can be used as an objective function in its own right, yielding texture generation, or as a regularizer in the inversion problem, yielding style transfer.

Algorithm 1 Stochastic gradient descent for pre-image

Require: Given the objective function $E(\cdot)$ and the learning rate η_0

$G_0 \leftarrow 0$, $\mu_0 \leftarrow 0$
Initialize $\mathbf{x}_1$ to random noise
for $t = 1$ to T **do**
  $g_t \leftarrow \nabla E(\mathbf{x}_t)$ (using backprop)
  $G_t \leftarrow \rho G_{t-1} + g_t^2$ (component-wise)
  $\eta_t \leftarrow \frac{1}{\frac{1}{\eta_0} + \sqrt{G_t}}$ (component-wise)
  $\mu_t \leftarrow \rho \mu_{t-1} - \eta_t g_t$
  $\mathbf{x}_{t+1} \leftarrow \Pi_{B_+}(\mathbf{x}_t + \mu_t)$
end for

Balancing the loss and the regularizers Note that all the regularizers above have been carefully normalized to balance the different terms. Given reasonable reconstruction $\mathbf{x}$, all $\mathcal{R}_*(\mathbf{x})$ above will have comparable values around unity. This normalization, though simple, makes a very significant difference. Without it we need to carefully tune parameters across different representation types. On the contrary in all our experiments we use one set of values: $C = 1$, $\alpha = 6$, $\beta = 2$, $B = 80$, $B_+ = 2B$, and $V = B/6.5$, unless otherwise noted.

5.2.3 Optimization

Finding a minimizer of the objective eq. (5.1) may seem difficult as most representations Φ are non-convex; in particular, deep representations are a composition of several non-linear layers. Nevertheless, simple gradient descent (GD) procedures have been shown to be very effective in *learning* such models from data, which is arguably an even harder task. In practice, a variant of GD was found to result in low error.

Algorithm. The algorithm, whose pseudocode is given in Algorithm 1, is a variant of AdaGrad [29]. Like in AdaGrad, our algorithm automatically adapts the learning rate of individual components of the vector $\mathbf{x}_t$ by scaling it by the inverse of the accumulated squared gradient G_t. Similarly to AdaDelta [155], however, it accumulates gradients only in a short temporal window, using the momentum coefficient $\rho = 0.9$. The gradient, scaled by the adaptive learning rate $\eta_t g_t$, is accumulated into a momentum vector μ_t with

the same factor ρ. The momentum is then summed to the current reconstruction $\mathbf{x}_t$ and the result is projected back onto the feasible region $[-B_+, B_+]$.

Recently, Gatys *et al.* [42, 41] have used the L-BFGS-B algorithm [166] to optimize their texture/style loss eq. (5.8). We found that L-BFGS-B is indeed better than (S)GD for their problem of texture generation, probably due to the particular nature of the term eq. (5.8). However, preliminary experiments using L-BFGS-B for inversion did not show a significant benefit, so for simplicity we consider (S)GD-based algorithms in this chapter.

The only parameters of Algorithm 1 are the initial learning rate η_0 and the number of iterations T. These are discussed next.

Initial learning rate η_0. We set the initial learning rate heuristically by considering only $\mathcal{R}_\alpha(x)$. To get some intuition, let $\mathbf{x}$ be a monochrome grayscale image with pixel intensity B (B was defined in eq. (5.5)). One update of gradient descent on a single pixel assigns it to $B - \eta_0\alpha/B$. To minimize $\mathcal{R}_\alpha(x)$ in this one step, we require

$$0 = B - \eta_0\alpha/B \implies \eta_0 = B^2/\alpha$$

Using this intuition we set,

$$\eta_0 = 0.01\frac{B^2}{\alpha}$$

Number of iterations T. Algorithm 1 is run for $T = 300$ iterations. When jittering is used as a regularizer, we found it beneficial to eventually disable it and run the algorithm for a further 50 iterations, $T = 350$ in total, after reducing the learning rate tenfold. This fine tuning does not change the results qualitatively, but for inversion it slightly improves the squared L^2 error; thus it is not applied in caricaturization and activation maximization.

The cost of running Algorithm 1 is dominated by the cost of computing the derivative of the representation function, usually by back-propagation in a deep neural network. By comparison, the cost of computing the derivative of the regularizers and the cost of the gradient update are negligible. This also means that the algorithm runs faster for shallower representations and slower for deeper ones; on a CPU, it may in practice take only a few seconds to visualize shallow layers in a deep network and a few minutes

for deep ones. GPUs can accelerate the algorithm by an order of magnitude or more. Another simple speedup is to stop the algorithm earlier; here using 300-350 iterations is a conservative choice that works for all representation types and visualizations we tested.

5.3 Representations

In this section, the image representations studied in the chapter - dense SIFT, HOG, and several reference deep CNNs, are described. It is also shown how DSIFT and HOG can be implemented in a standard CNN framework, which simplifies the computation of their derivatives as required by the algorithm of section 5.2.3.

5.3.1 Classical Representations

The *histograms of oriented gradients* are one of the best known family of "classical" computer vision features popularized by Lowe in [87] with the SIFT descriptor. Here we consider two densely-sampled versions [102], namely DSIFT (Dense SIFT) and HOG [17]. In the remainder of this section these two representations are reformulated as CNNs. This clarifies the relationship between SIFT, HOG, and CNNs in general and helps implement them in standard CNN toolboxes for experimentation. The DSIFT and HOG implementations in the VLFeat library [135] are used as numerical references. These are equivalent to Lowe's [87] SIFT and the DPMv5 HOG [35, 47].

SIFT and HOG involve: computing and binning image gradients, pooling binned gradients into cell histograms, grouping cells into blocks, and normalizing the blocks. Let us denote by $\mathbf{g}$ the image gradient at a given pixel and consider binning this into one of K orientations (where $K = 8$ for SIFT and $K = 18$ for HOG). This can be obtained in two steps: directional filtering and non-linear activation. The k^{th} directional filter is $G_k = u_{1k}G_x + u_{2k}G_y$ where

$$\mathbf{u}_k = \begin{bmatrix} \cos \frac{2\pi k}{K} \\ \sin \frac{2\pi k}{K} \end{bmatrix}, \quad G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_y = G_x^\top.$$

The output of a directional filter is the projection $\langle \mathbf{g}, \mathbf{u}_k \rangle$ of the gradient along direction $\mathbf{u}_k$. This is combined with a non-linear activation function to assign gradients to histogram

AlexNet			VGG-M			VGG-VD-16		
Name	Size	Stride	Name	Size	Stride	Name	Size	Stride
conv1	11	4	conv1	7	2	conv1_1	3	1
relu1	11	4	relu1	7	2	relu1_1	3	1
						conv1_2	5	1
						relu1_2	5	1
norm1	11	4	norm1	7	2			
pool1	19	8	pool1	11	4	pool1	6	2
conv2	51	8	conv2	27	8	conv2_1	10	2
relu2	51	8	relu2	27	8	relu2_1	10	2
						conv2_2	14	2
						relu2_2	14	2
norm2	51	8	norm2	27	8			
pool2	67	16	pool2	43	16	pool2	16	4
conv3	99	16	conv3	75	16	conv3_1	24	4
relu3	99	16	relu3	75	16	relu3_1	24	4
						conv3_2	32	4
						relu3_2	32	4
						conv3_3	40	4
						relu3_3	40	4
						pool3	44	8
conv4	131	16	conv4	107	16	conv4_1	60	8
relu4	131	16	relu4	107	16	relu4_1	60	8
						conv4_2	76	8
						relu4_2	76	8
						conv4_3	92	8
						relu4_3	92	8
						pool4	100	16
conv5	163	16	conv5	139	16	conv5_1	132	16
relu5	163	16	relu5	139	16	relu5_1	132	16
						conv5_2	164	16
						relu5_2	164	16
						conv5_3	196	16
						relu5_3	196	16
pool5	195	32	pool5	171	32	pool5	212	32
fc6	355	32	fc6	331	32	fc6	404	32
relu6	355	32	relu6	331	32	relu6	404	32
fc7	355	32	fc7	331	32	fc7	404	32
relu7	355	32	relu7	331	32	relu7	404	32
fc8	355	32	fc8	331	32	fc8	404	32
prob	355	32	prob	331	32	prob	404	32

Table 5.1: CNN architectures. Structure of the AlexNet, VGG-M, VGG-VD-16 CNNs giving '**Size**': Receptive field sizes, '**Stride**': strides between feature samples, both in pixels.

elements h_k. DSIFT uses bilinear orientation assignment, given by

$$h_k = \|\mathbf{g}\| \max\left\{0, 1 - \frac{K}{2\pi}\cos^{-1}\frac{\langle \mathbf{g}, \mathbf{u}_k\rangle}{\|\mathbf{g}\|}\right\},$$

whereas HOG (in the DPMv5 variant) uses hard assignment.

$$h_k = \|\mathbf{g}\|\mathbf{1}\left[\langle \mathbf{g}, \mathbf{u}_k\rangle > \|\mathbf{g}\|\cos\pi/K\right]$$

Filtering is a standard CNN operation but these activation functions are not. While their implementation is simple, an interesting alternative is to approximate bilinear orientation assignment by using the activation function:

$$\begin{aligned} h_k &\approx \|\mathbf{g}\| \max\left\{0, \frac{1}{1-a}\frac{\langle \mathbf{g}, \mathbf{u}_k\rangle}{\|\mathbf{g}\|} - \frac{a}{1-a}\right\} \\ &\propto \max\left\{0, \langle \mathbf{g}, \mathbf{u}_k\rangle - a\|\mathbf{g}\|\right\}, \quad a = \cos 2\pi/K. \end{aligned}$$

This activation function is the standard ReLU operator modified to account for the norm-dependent offset $a\|\mathbf{g}\|$. While the latter term is still non-standard, this indicates that a close approximation of binning can be achieved in standard CNN architectures.

The next step is to pool the binned gradients into cell histograms using bilinear spatial pooling, followed by extracting blocks of 2×2 (HOG) or 4×4 (SIFT) cells. Both operations can be implemented by banks of linear filters. Cell blocks are then l^2 normalized, which is a special case of the standard local response normalization layer. For HOG, blocks are further decomposed back into cells, which requires another filter bank. Finally, the descriptor values are clamped from above by applying $y = \min\{x, 0.2\}$ to each component, which can be reduced to a combination of linear and ReLU layers.

The conclusion is that approximations to DSIFT and HOG can be implemented with conventional CNN components plus the non-conventional gradient norm offset. However, all the filters involved are much sparser and simpler than the generic 3D filters in learned CNNs. The above analysis was meant to gain some intuition about DSIFT, HOG and CNNs. In the rest of the chapter, we will use exact CNN equivalents of DSIFT and HOG, using modified or additional CNN components as needed.[2] These CNNs are numerically indistinguishable from the VLFeat reference implementations,

[2]This requires addressing a few more subtleties. Please see files `dsift_net.m` and `hog_net.m` in the public source code for details. `https://github.com/aravindhm/nnpreimage/`

but, true to their CNN nature, allow computing the feature derivatives as required by the algorithm of section 5.2.3.

5.3.2 Deep Convolutional Neural Networks

The first CNN model considered in this chapter is **AlexNet**. Due to its popularity, we use the implementation that ships with the Caffe framework [70], which closely reproduces the original network by Krizhevsky *et al.* [76]. Occasionally, we also consider the **CaffeRef**, a network similar to AlexNet that also comes with Caffe. This and many other similar networks alternate the following computational building blocks: linear convolution, ReLU, spatial max-pooling, and local response normalization. Each such block takes as input a d-dimensional image and produces as output a k-dimensional one. Blocks can additionally pad the image (with zeros for the convolutional blocks and with $-\infty$ for max pooling) or subsample the data. The last several layers are deemed "fully connected" as the support of the linear filters coincides with the size of the image; however, they are equivalent to filtering layers in all other respects. Table 5.1 (left) details the structure of AlexNet.

The second network is the **VGG-M** model from [13]. The structure of VGG-M (table 5.1 – middle) is very similar to AlexNet, with the following differences: it includes a significantly larger number of filters in the different layers, filters at the beginning of the network are smaller, and filter strides (sub-sampling) are reduced. While the network is slower than AlexNet, it also performs much better on the ImageNet ILSVRC 2012 data.

The last network is the **VGG-VD-16** model from [121]. VGG-VD-16 is also similar to AlexNet, but with more substantial changes compared to VGG-M (table 5.1 – right). Filters are very narrow (3×3) and very densely sampled. There are no normalization layers. Most importantly, the network contains many more intermediate convolutional layers. The resulting model is very slow, but very powerful.

All pre-trained models are implemented in the MatConvNet framework [136] and are publicly available at `http://www.vlfeat.org/matconvnet/pretrained`.

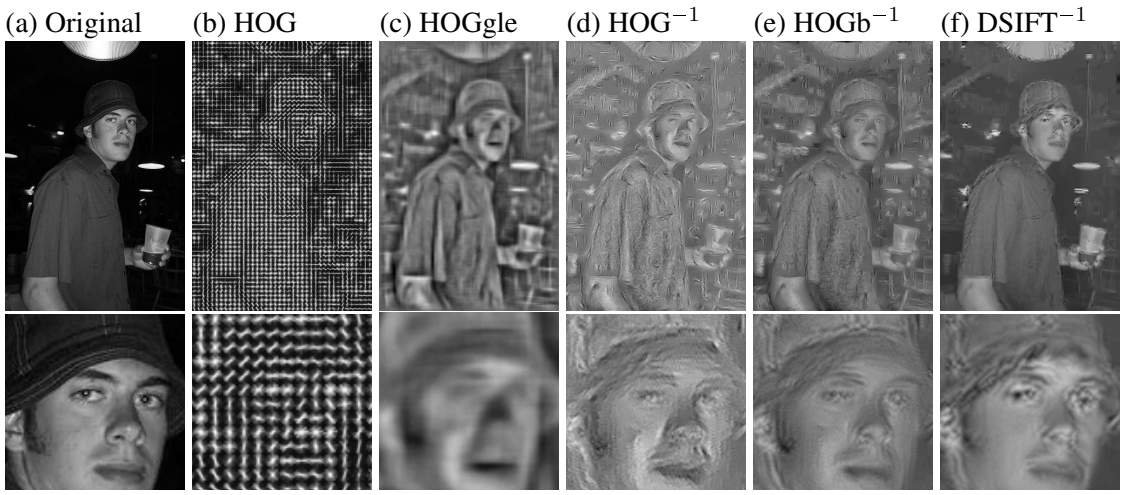


Figure 5.4: Reconstruction quality of different representation inversion methods, applied to HOG and DSIFT. HOGb denotes HOG with bilinear orientation assignments. From left to right: (a) The original probe image, (b) HOG-glyph, (c) HOGgle [137] inversion, (d-f) Inversions using our method. This image is best viewed on screen.

5.4 Visualization by Inversion

The experiments in this section apply the visualization by inversion method to both classical (section 5.4.1) and CNN (section 5.4.2) representations. As detailed in section 5.2.1, for inversion, the objective function eq. (5.1) is set up to minimize the L^2 distance eq. (5.2) between the representation $\Phi(\mathbf{x})$ of the reconstructed image $\mathbf{x}$ and the representation $\Phi_0 = \Phi(\mathbf{x}_0)$ of a reference image $\mathbf{x}_0$.

Importantly, the optimization starts by initializing the reconstructed image to random *independent and identically distributed (IID)* noise such that *the only information available to the algorithm is the code* Φ_0. When started from different random initializations, the algorithm is expected to produce different reconstructions. This is partly due to the local nature of the optimization, but more fundamentally to the fact that representations are designed to be invariant to nuisance factors. Hence, images with irrelevant differences should have the same representation and should be considered equally valid reconstruction targets. In fact, it is by observing the differences between such reconstructions that we can obtain insights into the nature of the representation's invariances.

Due to their intuitive nature, it is not immediately obvious how visualizations should be assessed quantitatively. Here we do so from multiple angles. The first is to test

Descriptor-Method	HOG-HOGle	HOG-our	HOGb-our	DSIFT-our
Error (%)	60.1 ± 2.3	36.6 ± 3.4	11.6 ± 0.9	9.4 ± 1.7

Table 5.2: Average reconstruction error of different representation inversion methods, applied to HOG and DSIFT. HOGb denotes HOG with bilinear orientation assignments. The error bars show the 95% confidence interval for the mean.

whether the algorithm successfully attains its goal of reconstructing an image $\mathbf{x}$ that has the desired representation $\Phi(\mathbf{x}) = \Phi_0$. In section 5.4.1 and section 5.4.2 this is tested in terms of the relative reconstruction error of eq. (5.2). Furthermore, in section 5.4.2 it is also verified for CNNs whether the reconstructed and original representations have the same "meaning", in the sense that they are mapped to the same class label. Note that such tests assess the reconstruction quality in *feature space* rather than in *image space*. This is an important point: as noted above, we are not interested in recovering an image $\mathbf{x}$ which is perceptually similar to the reference image $\mathbf{x}_0$; rather, in order to study the invariances of the representation Φ, we would like to recover an image $\mathbf{x}$ that differs from $\mathbf{x}_0$ but has the same representation. Measuring the difference in feature space is therefore appropriate. Finally, the effect of regularization is assessed empirically, via user assessment (section 5.4.2.2), to check whether the proposed notion of naturalness does in fact improve the interpretability of the visualizations.

5.4.1 Inverting Classical Representations: SIFT and HOG

In this section the visualization by inversion method is applied to the HOG and DSIFT representations.

Implementation details: The parameter C in eq. (5.1), trading off regularization and feature reconstruction fidelity, is set to 100 unless noted otherwise. Jitter is not used and the other parameters are set as stated in section 5.2.2. HOG and DSIFT cell sizes are set to 8 pixels.

Reconstruction quality: Based on the discussion above, the reconstruction quality is assessed by reporting the normalized reconstruction error (eq. (5.2)), averaged over the first 100 images in the ILSVRC 2012 validation set [117]. The closest alternative

to our inversion method is HOGgle, a technique introduced by Vondrick *et al.* [137] for visualizing HOG features. The HOGgle code is publicly available from the authors' website and is used throughout these experiments. HOGgle is pre-trained to invert the UoCTTI variant of HOG, which is numerically equivalent to the CNN-HOG network of section 5.3, which allows us to compare algorithms directly.

Compared to our method, HOGgle is faster (2-3s vs. 60s on the same CPU) but not as accurate, as is apparent qualitatively (fig. 5.4.c vs. d) and quantitatively (60.1% vs. 36.6% reconstruction error, see table 5.2). Vondrick *et al.* did propose a direct optimization baseline similar to eq. (5.1), but found that it did not perform better than HOGgle. This demonstrates the importance of the choice of regularizer and of the ability to compute the derivative of the representation analytically in order to implement optimization effectively. In terms of speed, an advantage of optimizing eq. (5.1) is that it can be switched to use GPU code immediately given the underlying CNN framework; doing so results in a ten-fold speed-up.

Representation comparison: Different representations are easier or harder to invert. For example, modifying HOG to use bilinear gradient orientation assignments as in SIFT (section 5.3) significantly reduces the reconstruction error (from 36.6% down to 11.5%) and improves the reconstruction quality (fig. 5.4.e). More remarkable are the reconstructions obtained by inverting DSIFT: they are quantitatively similar to HOG with bilinear orientation assignment, but produce significantly more detailed images (fig. 5.4.f). Since HOG uses a finer quantization of the gradient compared to SIFT but otherwise the same cell size and sampling, this result can be imputed to the stronger normalization in HOG that evidently discards more visual information than in SIFT.

5.4.2 Inverting CNNs

In this section the visualization by inversion method is applied to representative CNNs: AlexNet, VGG-M, and VGG-VD-16.

5.4.2.1 Implementation Details

The jitter amount T is set to the integer closest to a quarter of the *stride* of the representation; the stride is the step in the receptive field of representation components when stepping through spatial locations. Its value is given in table 5.1. The other parameters are set as stated in section 5.2.2.

The parameter C in eq. (5.1) is set to one of 1, 20, 100 or 300. Based on the analysis below, unless otherwise specified, visualizations use the following values: For AlexNet and VGG-M, we choose $C = 300$ up to relu3, $C = 100$ up to relu4, $C = 20$ up to relu5, and $C = 1$ for the remaining layers. For VGG-VD we use $C = 300$ up to conv4_3, $C = 100$ for conv5_1, $C = 20$ up to conv5_3, and $C = 1$ onwards.

5.4.2.2 Reconstruction Quality

The reconstruction accuracy is assessed in three ways: reconstruction error, consistency with respect to different random initializations, and classification consistency.

Reconstruction error: Similar to section 5.4.1, the reconstruction error eq. (5.2) is averaged over the first 100 images in the ILSVRC 2012 validation set [117] (these images were not used to train the CNNs). The experiment is repeated for all the layers of AlexNet and for different values of the parameter C to assess its effect. The resulting average errors are reported in fig. 5.5 (top-left panel).

CNNs such as AlexNet are significantly larger and deeper than the CNN implementations of HOG and DSIFT. Therefore, it seems that the inversion problem should be considerably harder for them. Instead, comparing the results in fig. 5.5 to the ones in table 5.2 indicates that CNNs are, in fact, not much more difficult to invert than HOG. In particular, for a sufficiently large value of C, the reconstruction error can be maintained in the range 10–20%, including for the deepest layers. Therefore, the non-linearities in the CNN seem to be rather benign, which could explain why SGD can learn these models successfully. Using a stronger regularization (small C) significantly deteriorates the quality of the reconstructions from earlier layers of the network. At the same time, it has little to no effect on the reconstruction quality of deeper layers. Since, as verified

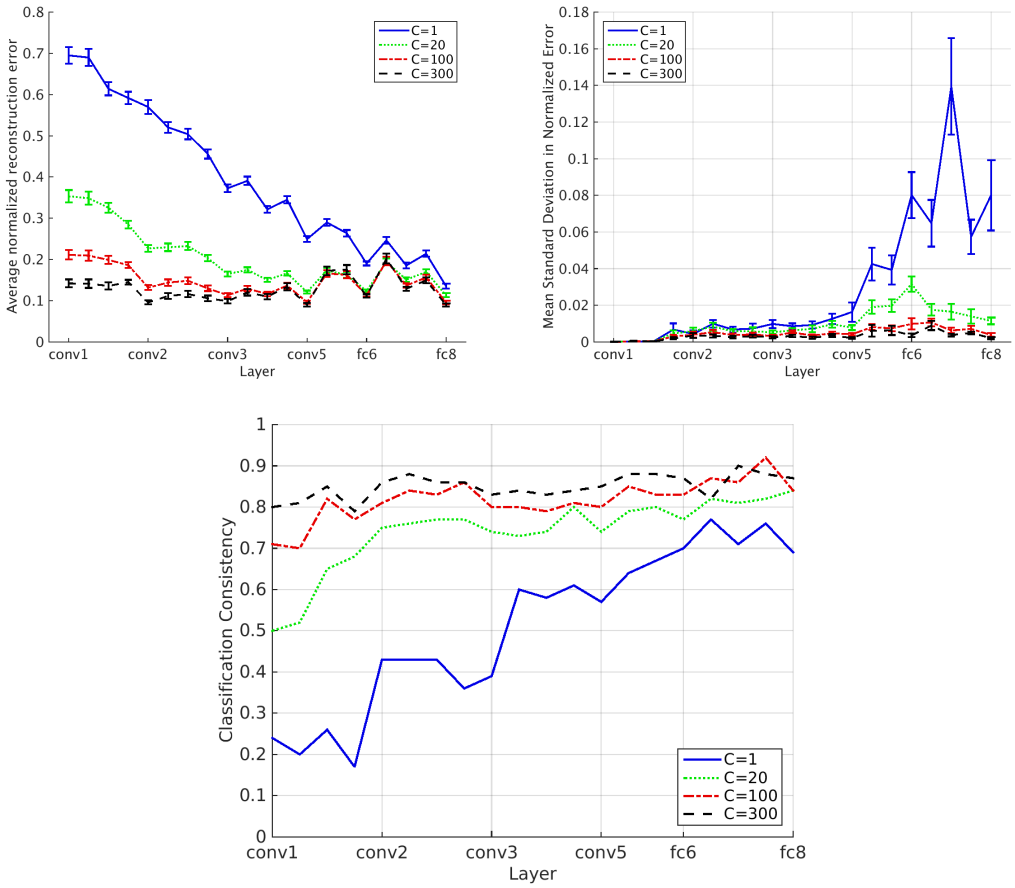


Figure 5.5: *Quality of CNN inversions.* The plots report three reconstruction quality indices for the CNN inversion method applied to different layers of AlexNet and for different reconstruction fidelity strengths C. The top-left plot shows the average reconstruction error averaged using 100 ILSVRC validation images as reference images (the error bars show the 95% confidence interval for the mean). The top-right plot reports the standard deviation of the reconstruction error obtained from 24 different random initializations per reference image, and then averaged over 24 such images. The bottom plot reports classification consistency: The fraction of reconstructions that are associated by the CNN to the same class as their reference image.

below, a strong regularization significantly improves the interpretability of resulting pre-images, it should be used for these layers.

Consistency of multiple pre-images: As explained earlier, different random initializations are expected to result in different reconstructions. However, this diversity should reflect genuine representation ambiguities and invariances rather than the inability of the local optimization method to escape bad local optima. To verify that this is the case, the standard deviation of the reconstruction error eq. (5.2) is computed from 24 different reconstructions obtained from the same reference image and 24 different initializations.

The experiment is repeated using, as reference, the first 24 images in the ILSVRC 2012 validation dataset and the average standard deviation of the reconstruction errors is reported in fig. 5.5 (top-right panel). This figure shows that, for the values of C except $C = 1$, all pre-images have very similar reconstruction errors, with standard deviation of around 0.02 or less. Thus in all but the very deep layers all pre-images can be treated as equally useful from the viewpoint of reconstruction error. In the next paragraph, we show that even for very deep layers, pre-images are substantially equivalent from the viewpoint of classification consistency.

Classification consistency: One question that may arise is whether a reconstruction error of 20%, or even 10%, is sufficiently small to validate the visualizations. To answer this question, fig. 5.5 (bottom panel) reports the *classification consistency* of the reconstructions. Here "classification consistency" is the fraction of reconstructed pre-images $\mathbf{x}$ that the CNN associates with the same class label as the reference image $\mathbf{x}_0$. This value, which would be equal to 1 for perfect reconstructions, measures whether imperfections in the reconstruction process are small enough to not affect the "meaning" of the image from the viewpoint of the CNN.

As it may be expected, classification consistency results show a trend similar to the reconstruction error, where better reconstructions are obtained for small amounts of regularization for the shallower layers, whereas deep layers can afford much stronger regularization. It is interesting to note that even visually odd inversions of deep layers such as those shown in fig. 1.1 or fig. 5.8 are classification consistent with their reference image, demonstrating the high degree of invariance in such layers. Finally, we note that, by choosing the correct amount of regularization, the classification consistency of the reconstructions can be kept above 0.8 in most cases, validating the visualizations.

Naturalness: One of our contributions is the idea that imposing even simple naturalness priors on the reconstructed images improves their interpretability. Since interpretability is a purely subjective attribute, in this section we conduct a small user study to validate this idea. Before that, however, we check whether regularization works as expected and produces images that are statistically closer to natural ones.

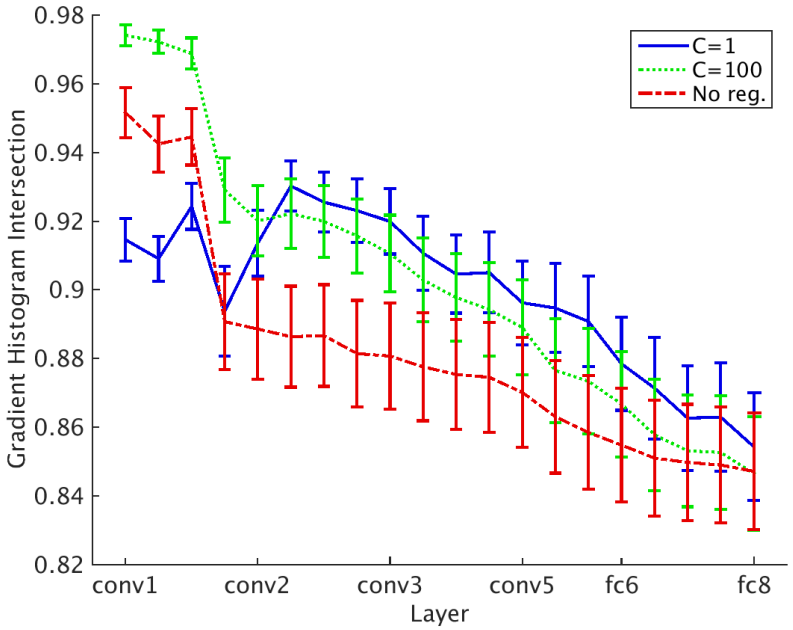


Figure 5.6: Mean histogram intersection similarity between the gradient histogram of the reference image $\mathbf{x}_0$ and gradient histogram of the computed pre-image $\mathbf{x}$; for different layers of AlexNet and values of the parameter C (only a few such values are reported to reduce clutter). Error bars show the 95% confidence interval of the mean histogram intersection similarity.

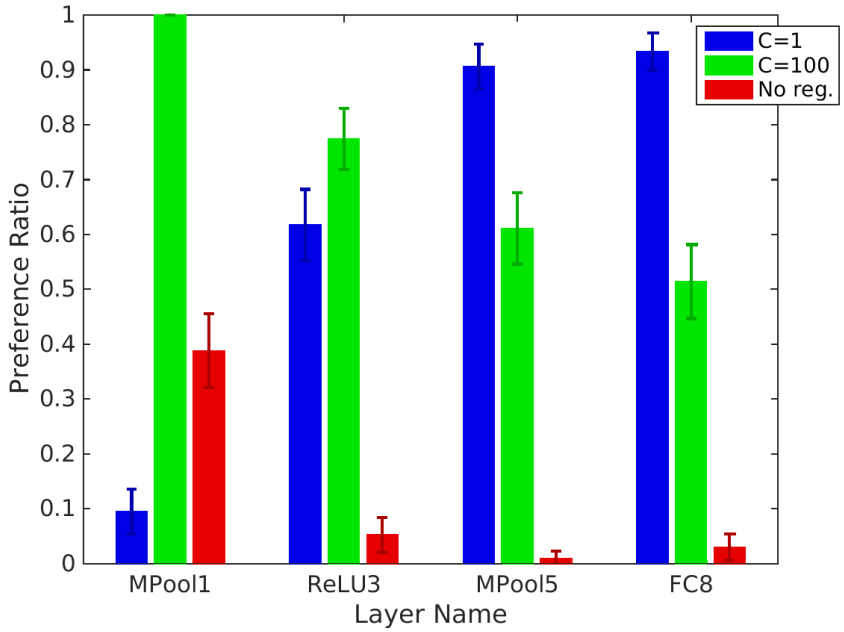


Figure 5.7: Fraction of times a certain regularization parameter value C was found to be more interpretable than one of the other two. Interpretability was assessed by users looking at pre-images from the layers of AlexNet.

Natural images are well known to have certain *statistical regularities*; for example, the magnitude of image gradients have an exponential distribution [59]. Here we check whether regularized pre-images are more "natural" by comparing them to natural images using such statistics. To do so, we compute the histogram of gradient magnitudes for the 100 ImageNet ILSVRC reference images used above and for their AlexNet inversions, for different values of the parameter C. Then the original and reconstructed histograms are compared using histogram intersection and their similarity is reported in fig. 5.6. A small amount of regularization is clearly preferable for shallow layers, and a stronger amount is clearly better for intermediate ones. However, the difference is not that significant for the deepest layers, which are therefore best analyzed in terms of their interpretability.

To test **interpretabilty**, inversions were obtained using the first 25 ILSVRC 2012 validation images as reference. Inversions were obtained from the mpool1, relu3, mpool5, and fc8 layers of AlexNet for three regularizations strengths: (a) no regularization ($C = \infty$), (b) weak regularization ($C = 100$), and (c) strong regularization ($C = 1$). Thus we have three pre-images per layer per reference image. In a user study, each subject was shown two randomly picked regularization settings for a layer and reference image. Each subject was asked to select the image that was more interpretable ("whose content is more easily understood"). We conducted this study with 13 subjects who were not familiar with this line of research, or not familiar with computer vision at all. The first five votes from each subject were discarded to allow them to become familiar with the task. The ordering of images and layers was randomized independently for each subject. Uniform random sampling between the three regularization strengths ensures that no regularization strength dominates the screen or even one side of the screen. Figure 5.7 shows the fraction of time a certain regularization strength was found to produce more interpretable results for a given AlexNet layer. Based on these results, at least a small amount of regularization is always preferable for interpretability. Furthermore, strong regularization is highly desirable for very deep layers.

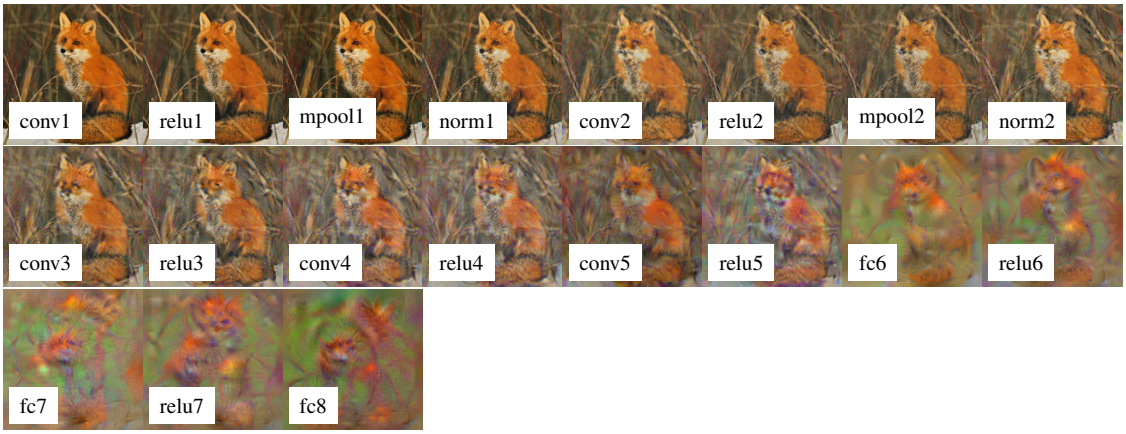


Figure 5.8: AlexNet inversions (all layers) from the representation of the "red fox" image obtained from each layer of AlexNet.

Figure 5.9: VGG-M inversions (selected layers). This figure is best viewed in color.

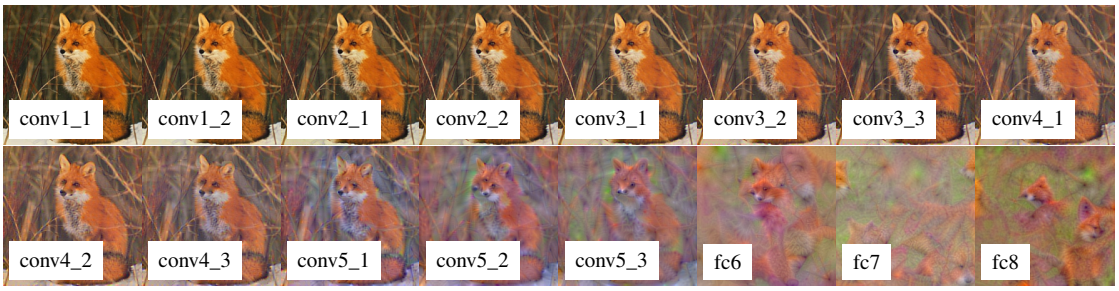


Figure 5.10: VGG-VD-16 inversions (selected layers). This figure is best viewed in color.

5.4.2.3 Inversion of Different Layers

Having established the legitimacy of the inversions, next we qualitatively study the reconstructions obtained from different layers of the three CNNs for a test image ("red fox"). In particular, fig. 5.8 shows the reconstructions obtained from each layer of AlexNet and fig. 5.9 and fig. 5.10 do the same for all the linear (convolutional and fully connected) layers of VGG-M and VGG-VD.

The progression is remarkable. The first few layers of all the networks compute a code of the image that is nearly exactly invertible. All the layers prior to the fully-connected ones preserve instance-specific details of the image, although with increasing fuzziness. The 4,096-dimensional fully connected layers discard more geometric as well as instance-

specific information as they invert back to a *composition of parts, which are similar but not identical to the ones found in the original image*.

Comparing different architectures, VGG-M reconstructions are sharper and more detailed than the ones obtained from AlexNet, as it may be expected due to the denser and higher dimensional filters used here (compare for example conv4 in fig. 5.8 and fig. 5.9). VGG-VD emphasizes these differences more. First, abstractions are achieved much more gradually in this architecture: conv5_1, conv5_2 and conv5_3 reconstructions resemble the reconstructions from conv5 in AlexNet and VGG-M, despite the fact that there are three times more intermediate layers in VGG-VD. Nevertheless, fine details are preserved more accurately in the deep layers in this architecture (compare for example, the nose and eyes of the fox in conv5 in VGG-M and conv5_1 – conv5_3 in VGG-VD).

Second, reconstructions from deep VGG-VD layers are often more zoomed in compared to other networks (see for example the "abstract art" and "monkey" reconstructions from fc7 in fig. 5.11 and, for activation maximization, in fig. 5.18). The preference of VGG-VD for large, detailed object occurrences may be explained by its ability to better represent fine-grained object details, such as textures.

5.4.2.4 Reconstruction Ambiguity and Invariances

Figure 5.11 examines the invariances captured by the VGG-VD codes by comparing multiple reconstructions obtained from several deep layers. A careful examination of these images reveals that the codes capture progressively larger deformations of the object. In the "spoonbill" image, for example, conv5_2 reconstructions show slightly different body poses, evident from the different leg configurations. In the "abstract art" test image, a close examination of the pre-images reveals that, while the texture statistics are preserved well, the instance-specific details are in fact completely different in each image: the location of the vertexes, the number and orientation of the edges, and the color of the patches are not the same at the same locations in different images. This case is also remarkable as the training data for VGG-VD, i.e. ImageNet ILSVRC train set, does not contain such patterns suggesting that these codes are indeed rather generic. Inversions from fc7 result in multiple copies of the object/parts at different positions

and scales for the "spoonbill" and "monkey" cases. For the "monkey" and "abstract art" cases, inversions from fc7 appear to result in slightly magnified versions of the pattern: for instance, the reconstructed monkey's eye is about 20% larger than in the original image; and the reconstructed patches in "abstract art" are about 70% larger than in the original image. The preference for reconstructing larger object scales seems to be typical of VGG-VD (see also fig. 5.18).

Note that all these reconstructions and the original images are very similar from the viewpoint of the CNN representation; we conclude in particular that the deepest layers find the original images and a number of scrambled parts to be equivalent. This may be considered as another type of natural confounder for CNNs alternative to those discussed in [96].

5.4.2.5 Reconstruction Biases

It is interesting to note that some of the inverted images have large green regions (for example see fig. 5.10 fc6 to fc8). This property is likely to be intrinsic to the networks and not induced, for example, by the choice of natural image prior, the effect of which is demonstrated in fig. 5.2 and fig. 5.3. The prior only encourages smoothness as it is equivalent (for $\beta = 2$) to penalizing high-frequency components of the reconstructed image. Importantly, the prior is applied to all color channels equally. When gradually removing the prior, random high-frequency components dominate and it is harder to discern a human-interpretable signal.

5.4.2.6 Inversion from Selected Representation Components

It is also possible to examine reconstructions obtained from subsets of neural responses in different CNN layers. Figure 5.12 explores the *locality* of the codes by reconstructing a central 5×5 patch of features in each layer. The regularizer encourages portions of the image that do not contribute to the neural responses to be switched off. The locality of the features is obvious in the figure; what is less obvious is that the effective receptive field of the neurons is in some cases significantly smaller than the theoretical one shown as a red box in the image.

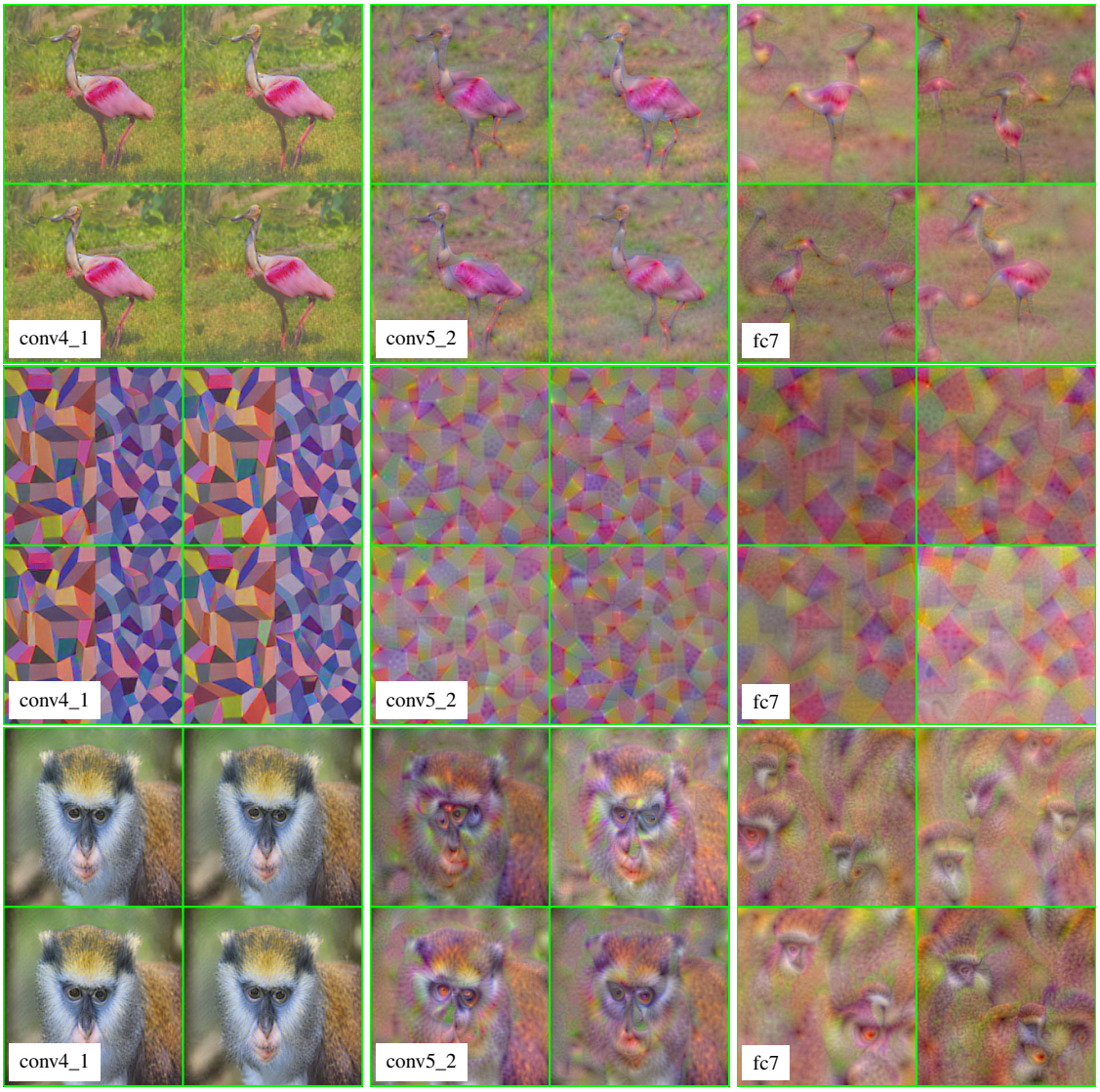


Figure 5.11: For three test images, "spoonbill", "abstract art", and "monkey", we generate four different reconstructions from layers conv4_1, conv5_2, and fc7 in VGG-VD. This figure is best seen in color.

Finally, fig. 5.13 reconstructs images from two different subsets of feature channels for CaffeNet. These subsets are induced by the fact that the first several layers (up to norm2) of the CaffeNet architecture are trained to have blocks of independent filters [76]. We invert these individually. Notice that one group is tuned towards color information, and the second one is tuned towards sharper edges and luminance components. Remarkably, this behavior emerges spontaneously in the learned network.

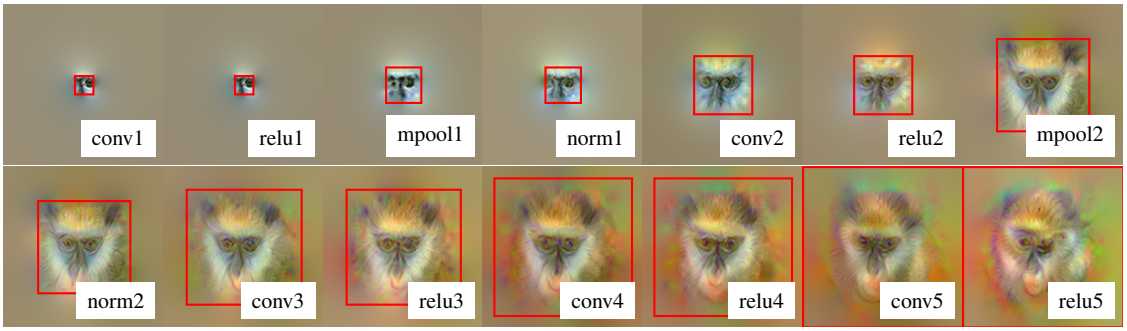


Figure 5.12: Reconstructions of the "monkey" image from a central 5×5 window of feature responses in the convolutional layers of CaffeRef. The red box marks the overall receptive field of the 5×5 window.

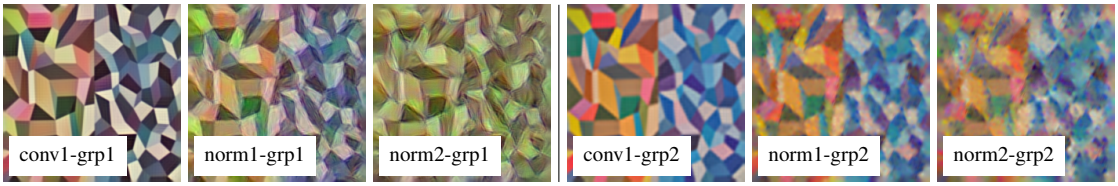


Figure 5.13: CNN neural streams. Reconstructions of the "abstract art" test image from either of the two neural streams in CaffeRef. This figure is best seen in color.

5.5 Visualization by Activation Maximization

In this section the activation maximization method is applied to classical and CNN representations. Recall that this visualization method optimizes a randomly initialized pre-image $\mathbf{x}$, so as to maximize the response of a single feature component $[\Phi(\mathbf{x})]_i$. We assume that the resulting image characterizes the collection of visual stimuli that the chosen neuron is selective for. Loosely speaking, while inversion visualizes what is "captured" by a representations, neuron maximization visualizes what a neuron would "like to see".

5.5.1 Classical Representations

We use activation maximization to visualize HOG templates. Let $\Phi_{\text{HOG}}(\mathbf{x})$ denote the HOG descriptor of a gray scale image $\mathbf{x} \in \mathbb{R}^{H \times W}$; a *HOG template* is a vector $\mathbf{w}$, usually learned by a SVM, that defines a scoring function $\Phi(\mathbf{x}) = \langle \mathbf{w}, \Phi_{\text{HOG}}(\mathbf{x}) \rangle$ for a particular object category. The function $\Phi(\mathbf{x})$ can be interpreted as a CNN consisting of the HOG CNN $\Phi_{\text{HOG}}(\mathbf{x})$ followed by a linear projection layer with parameter $\mathbf{w}$. The output of $\Phi(\mathbf{x})$ is a scalar, expressing the confidence that the image $\mathbf{x}$ contains the target object class.

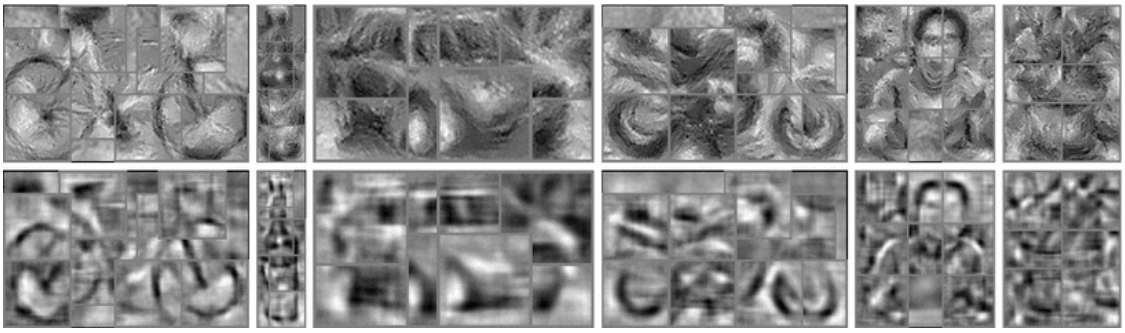


Figure 5.14: Visualization of DPMv5 HOG models using activation maximization (top) and HOGgle (bottom). Each model comprises a "root" filter overlaid with several part filters. From left to right: Bicycle (Component #2), Bottle (#5), Car (#4), Motorbike (#2), Person (#1), Potted Plant (#4).

In order to visualize the template $\mathbf{w}$ using activation maximization, the loss function $-\langle\Phi(\mathbf{x}), \Phi_0\rangle/Z$ of eq. (5.4) is plugged into the objective function eq. (5.1). Since in this case $\Phi(\mathbf{x})$ is a scalar function, the reference vector Φ_0 is also a scalar, which is set to 1. The normalization constant Z is set to

$$Z = M\rho^2 \tag{5.9}$$

where $\rho = \max\{H, W\}$ and M is an estimate of range of $\Phi(\mathbf{x})$, obtained as $M = \langle|\mathbf{w}|, \Phi_{\text{HOG}}(\mathbf{x})\rangle$ where $|\mathbf{w}|$ is the element wise absolute value of $\mathbf{w}$ and $\mathbf{x}$ is set to a white noise sample. This method is used to visualize the Deformable Part Models (DPMv5) [47] trained on the VOC2010 [31] dataset (Figure 5.14).

These visualizations are compared to the ones obtained in the analogous experiment by Vondrick *et al.* ([137] Fig. 14) using HOGgle. An important difference is that HOGgle does not perform activation maximization, but rather uses inversion and returns an approximate pre-image $\Phi^{-1}_{\text{HOG}}(\mathbf{w}_+)$, where $\mathbf{w}_+ = \max\{0, \mathbf{w}\}$ is the rectified template. Strictly speaking, inversion is not applicable here because the template $\mathbf{w}$ is *not* a HOG descriptor. In particular, $\mathbf{w}$ contains negative values which HOG descriptors do not contain. Even after rectification, there is usually no image such that $\Phi_{\text{HOG}}(\mathbf{x}) = \mathbf{w}_+$. On the other hand, activation maximization is principled because it works on top of the detector scoring function; in this manner it can correctly reflect the effect of both positive and negative values in $\mathbf{w}$.

The visualizations of five DPMs using activation maximization and HOGgle are shown in fig. 5.14. Note that the DPMs are hierarchical, and consist of a root object template and several higher-resolution part templates. For simplicity, each part is processed independently, but activation maximization could be applied to the composition of all the parts to remove the seams between them.

Compared to HOGgle, activation maximization reconstructs finer details, as can be noted for parts such as the bicycle wheel and bottle top. On the other hand, HOGgle reconstructions contain stronger and straighter edges (see for example the car roof). The latter may be a result of HOGgle using a restricted dictionary computed from natural images whereas our approach uses a more generic smoothness prior.

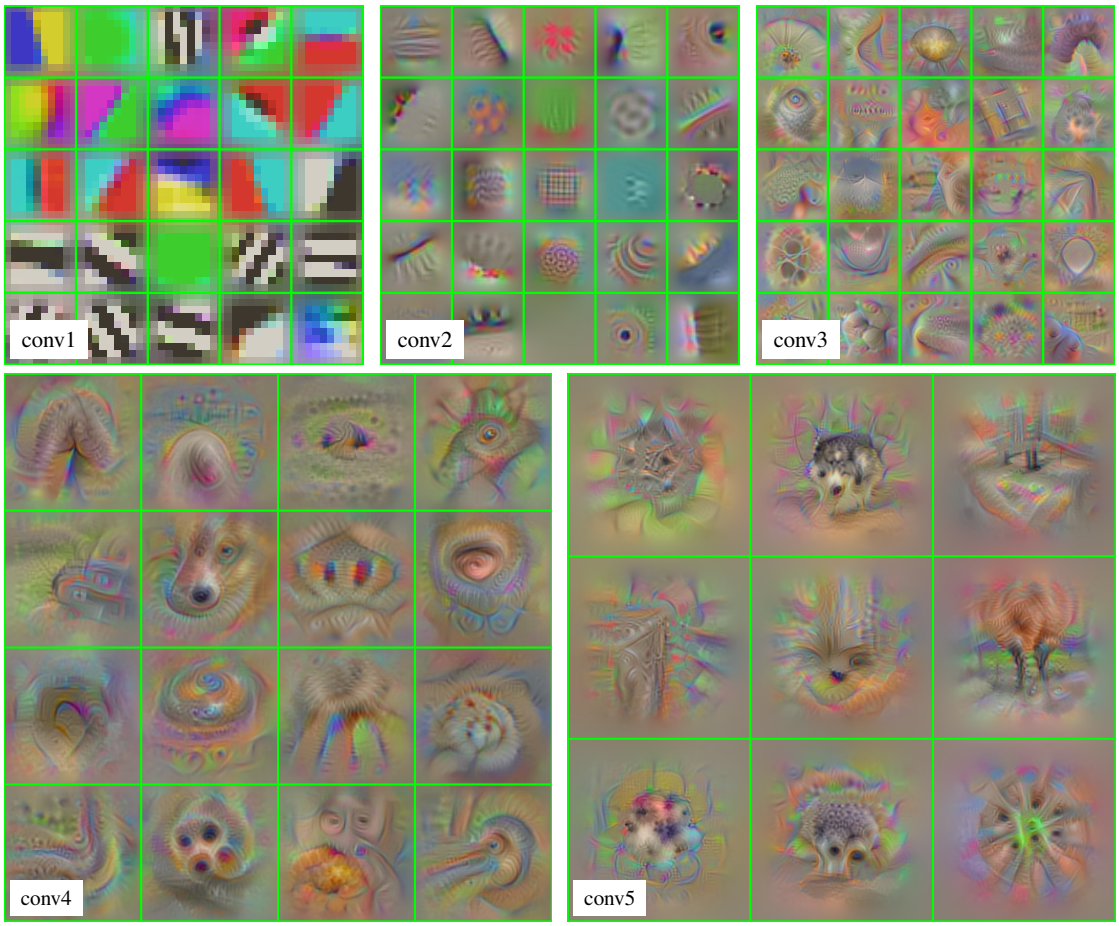


Figure 5.15: Activation maximization of the first filters of each convolutional layer in VGG-M.

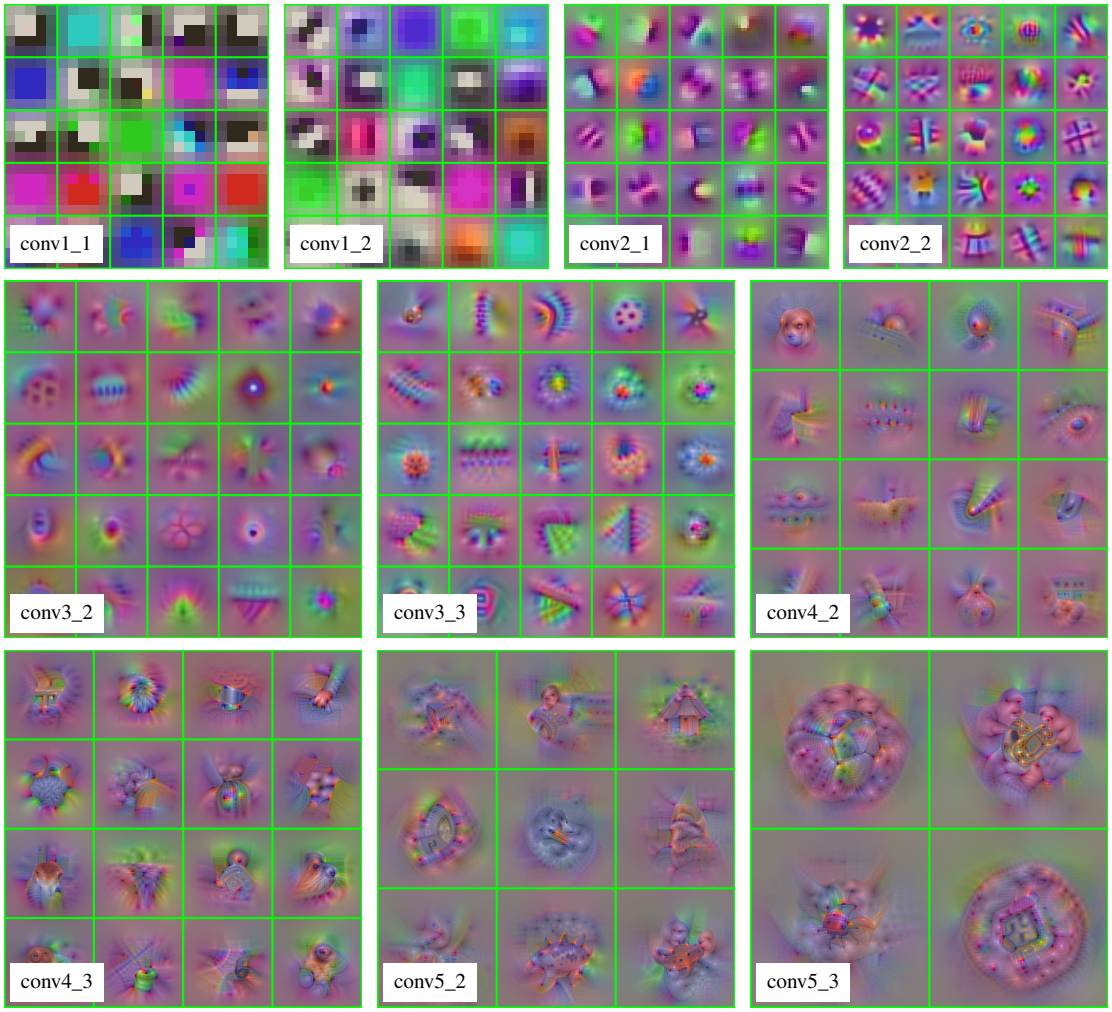
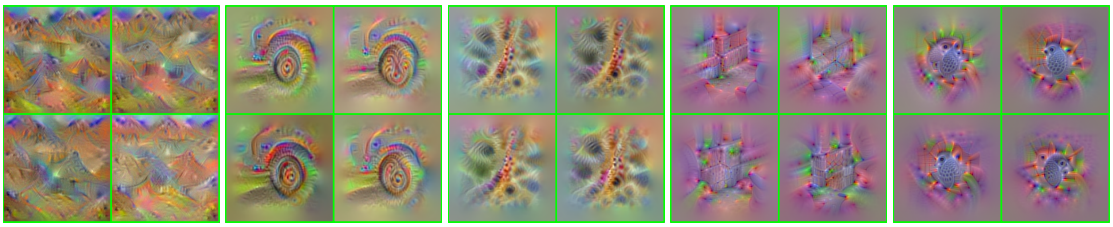


Figure 5.16: Activation maximization of the first filters for each convolutional layer in VGG-VD-16.



AlexNet	VGG-M	VGG-M	VGG-VD-16	VGG-VD-16
fc8 #971	conv4 #408	conv4 #392	conv5_2 #263	conv5_2 #26

Figure 5.17: Activation maximization with 4 different initializations.

5.5.2 CNN Representations

Next, the activation maximization method is applied to the study of deep CNNs. As before, the inner-product loss eq. (5.4) is used in the objective eq. (5.1), but this time the reference

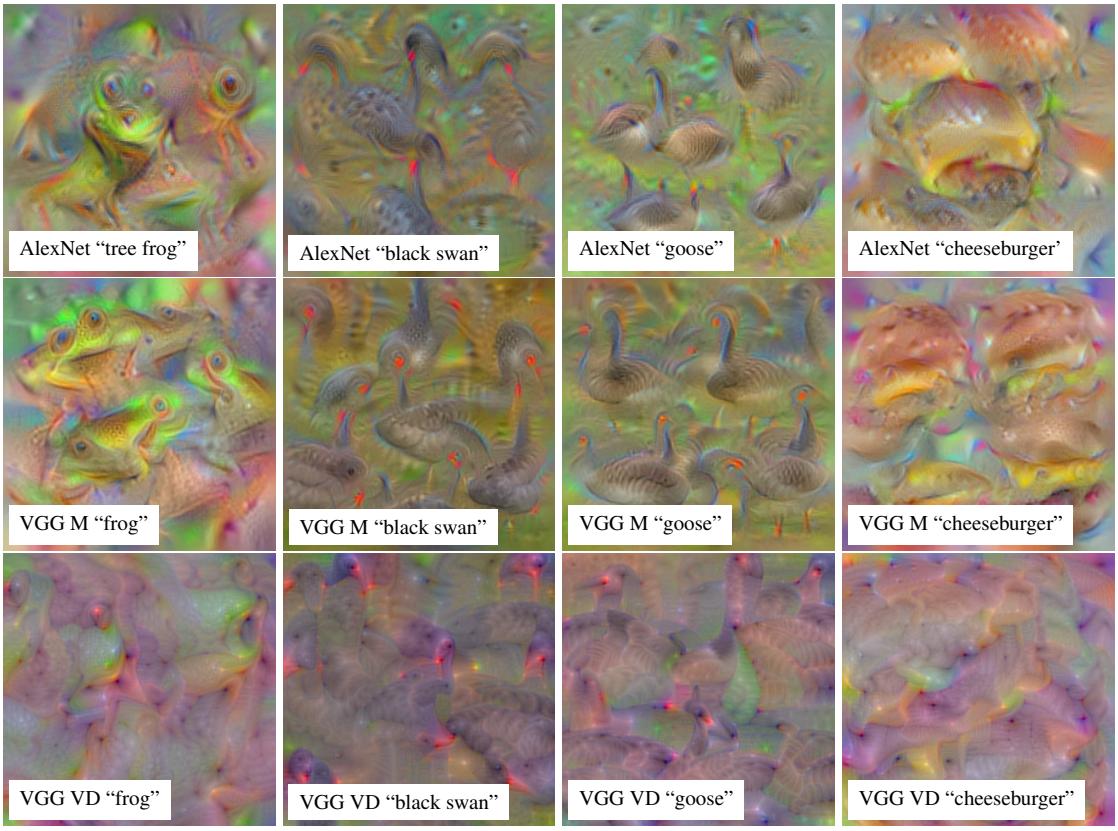


Figure 5.18: Activation maximization for the second to last layer of AlexNet, VGG-M, VGG-VD-16 for the classes “frog”, “black swan”, “goose”, and “vending machine”. The second to last layer codes directly for different classes, before softmax normalization.

vector Φ_0 is set to the one-hot indicator vector of the representation component being visualized. It was not possible to find an architecture independent normalization constant Z in eq. (5.4) as different CNNs have very different ranges of neuron output. Instead Z is calculated in the same way as eq. (5.9), $Z = M\rho^2$, where M is the maximum value achieved by the representation component in the ImageNet ILSVRC 2012 validation dataset and ρ the size of the component receptive field, as reported in table 5.1. As in section 5.4.2 the jitter amount T is set to a fourth of the feature-stride. All other parameters are set as described in section 5.2.2 (including $C = 1$).

Figure 5.15 shows the visual patterns obtained by maximally activating neurons in the convolutional layers of VGG-M. Similarly to [156] and [154], the complexity of the patterns increases substantially with depth. The first convolutional layer conv1 captures colored edges and blobs, but the complexity of the patterns generated by conv3, conv4 and conv5 is remarkable. While some of these patterns do evoke objects or object parts,

it remains difficult to associate with them a clear semantic interpretation. This is not entirely surprising given that the representation is distributed and activations may need to be combined to form a meaning. Experiments from AlexNet yielded entirely analogous if a little blurrier results. These are discussed in chapter 6.

Figure 5.16 shows the patterns obtained from VGG-VD. The complexity of the patterns build up more gradually than for VGG-M and AlexNet. Qualitatively, the complexity of the stimuli in conv5 in AlexNet and VGG-M seems to be comparable to conv4_3 and conv5_1 in VGG-VD. Conv5_2 and conv5_3 do appear to be significantly more complex, however. A second observation is that the reconstructed colors tend to be much more saturated, probably due to the lack of normalization layers in the architecture. Thirdly, we note that reconstructions contain significantly more fine-grained details, and in particular tiny blob-like structures, which probably activate very strongly the first very small filters in the network.

Figure 5.18 repeats the experiment from [121], more recently reprised by [154] and [93], and maximizes components of fc8 that correspond to a given class prediction. Four classes are considered: two similar animals (“black swan” and “goose”), a different one (“tree frog”), and an inanimate object (“cheeseburger”). We note that in all cases it is easy to identify several parts or even instances of the target object class. However, reconstructions are fragmented and scrambled, indicating that the representations are highly invariant to occlusions and pose changes. Secondly, reconstructions from VGG-M are considerably sharper than the ones obtained from AlexNet, as could be expected. Thirdly, VGG-VD-16 differs significantly from the other two architectures. Colors are more “washed out”, which we impute to the lack of normalization in the architecture as for fig. 5.16. Reconstructions tends to focus on much larger objects; for example, the network clearly captures the feather pattern of the bird as well as the rough skin of the frog.

Finally, fig. 5.17 shows multiple pre-images of a few representation components obtained by starting activation maximization from different random initializations. This is analogous to fig. 5.11 and is meant to probe the invariances in the representation.

The variation across the four pre-images is a mix of geometric and style transformations. For example, the second neuron appears to represent four different variants of a wheel of a vehicle.

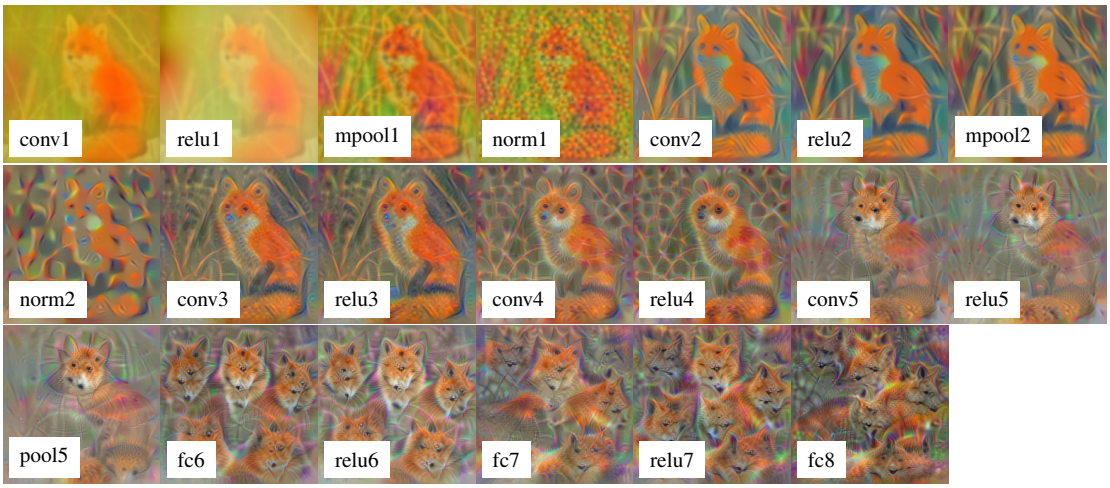


Figure 5.19: Caricatures of the "red fox" image obtained from the different layers in VGG-M.

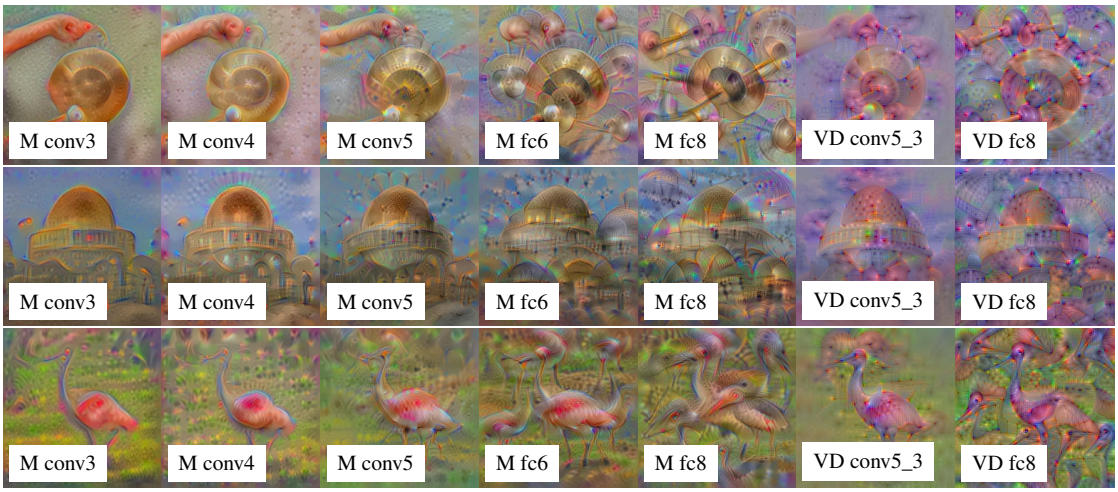


Figure 5.20: Caricatures of a number of test images obtained from the different layers of VGG-M (conv3, conv4, conv5, fc6, fc8) and VGG-VD (conv5_3 and fc8).

5.6 Visualization by Caricaturization

Our last visualization is inspired by Google's Inceptionism [93]. It is similar to activation maximization (section 5.5) and in fact uses the same formulation for eq. (5.1) with

the inner-product loss eq. (5.4). However, there are two key differences. First, the target mask is now set to

$$\Phi_0 = \max\{0, \Phi(\mathbf{x}_0)\} \tag{5.10}$$

where $\mathbf{x}_0$ is a reference image and the normalization factor Z is set to $\|\Phi_0\|^2$. Second, the optimization is started from the image $\mathbf{x}_0$ itself.

The idea of this visualization is to exaggerate any pattern in $\mathbf{x}_0$ that activates neurons in the representation $\Phi(\mathbf{x}_0)$, hence creating a "caricature" of the image according to this model. Furthermore, differently from activation maximization, this visualization works with combinations of multiple activations instead of individual ones.

Figure 5.19 shows the caricatures of the "red-fox" image obtained from the different layers of VGG-M. Applied to the first block of layers, the procedure simply saturates the color. Conv2 appears to be tuned to long, linear structures, conv4 to round ones, and conv5 to the head (part) of the fox. The fully connected layers generate mixtures of fox heads, including hallucinating several in the background, as already noted in [93]. Figure 5.20 shows the caricatures obtained from selected layers of VGG-M and VGG-VD, with similar results.

5.7 Discussion

We have presented three visualization tools (inversion, activation maximization, and caricaturization) which were made interpretable using natural image priors. We used them to probe and compare standard classical representations, and CNNs.

We compared against HOGgle for HOG inversion and obtained better quantitative results. The most important observations, however, emerged from an analysis of the visualizations obtained from deep CNNs; some of these are: that photometrically accurate information is preserved deep down in CNNs, that even very deep layers contain instance-specific information about objects, that intermediate convolutional layers capture local invariances to pose and fully connected layers to large variations in the object's layout, that

individual CNN components code for complex but, for the most part, not semantically-obvious patterns, and that different CNN layers appear to capture different types of structures in images, from lines and curves to parts.

The robustness of our inversion method was established by (a) evaluating the residual error in representation space, (b) estimating the variance in this error and (c) checking whether the inversions are classified the same way as the representation being inverted. We also analyzed the interpretability of inversions across varying objective weights (C) using a user-study. We found that some amount of regularization always yields a more interpretable result than no regularization. Under the right choice of objective weight C, we can achieve low error, high classification consistency and a more interpretable results.

We shall apply these tools to visualize our self-supervised CPFS-CE model in the next chapter.

6

Visualizing Representations Self-Supervised Using Motion

This chapter applies the visualization suite of chapter 5 to the self-supervised CPFS-CE model trained using YFCC100m-1600k dataset in chapter 3. This model's architecture is very similar to a standard AlexNet. We therefore contrast it with visualizations of an ILSVRC-12 pre-trained AlexNet and a randomly initialized version of the model itself. We also explore the interpretable neurons in our model using the Network Dissection method [7].

6.1 Neuron Maximization

We use per-neuron activation maximization to visualize individual neurons in the various convolutional and fully connected layers. Fig. 6.1 and fig. 6.2 present the estimated optimal stimulus for each the first 25 neurons from the five convolutional layers, made interpretable using a natural image prior. They do so for all three networks - our motion pre-trained model, an ILSVRC-12 pre-trained Alexnet and a randomly initialized model. We observe abstract patterns in both pre-trained models that are not present in a random network, suggesting that the representation may capture concepts useful for general-purpose image analysis. The ILSVRC-12 pre-trained neurons vaguely resemble semantic objects, unlike the motion pre-trained neurons; except for conv5 - row 4, column 4 which

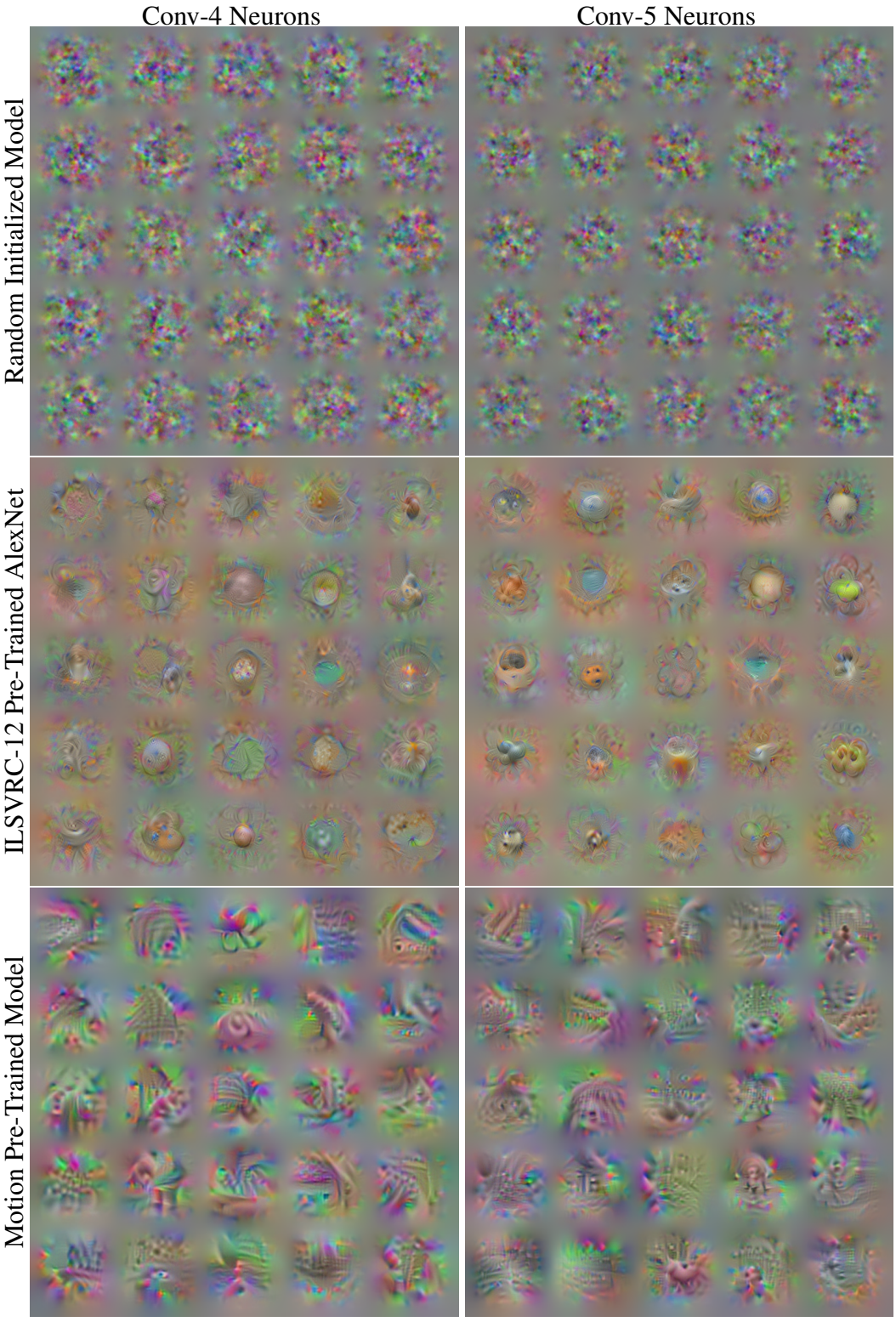


Figure 6.1: Neuron Maximization: Six grids are shown one for each model across two layers (Conv-4 and Conv-5). Each grid itself consists of 25 images. Each image in the grid corresponds to the estimated optimal excitatory stimulus for a neuron in the corresponding network. This visualization is obtained using the regularized neuron maximization method of chapter 5.

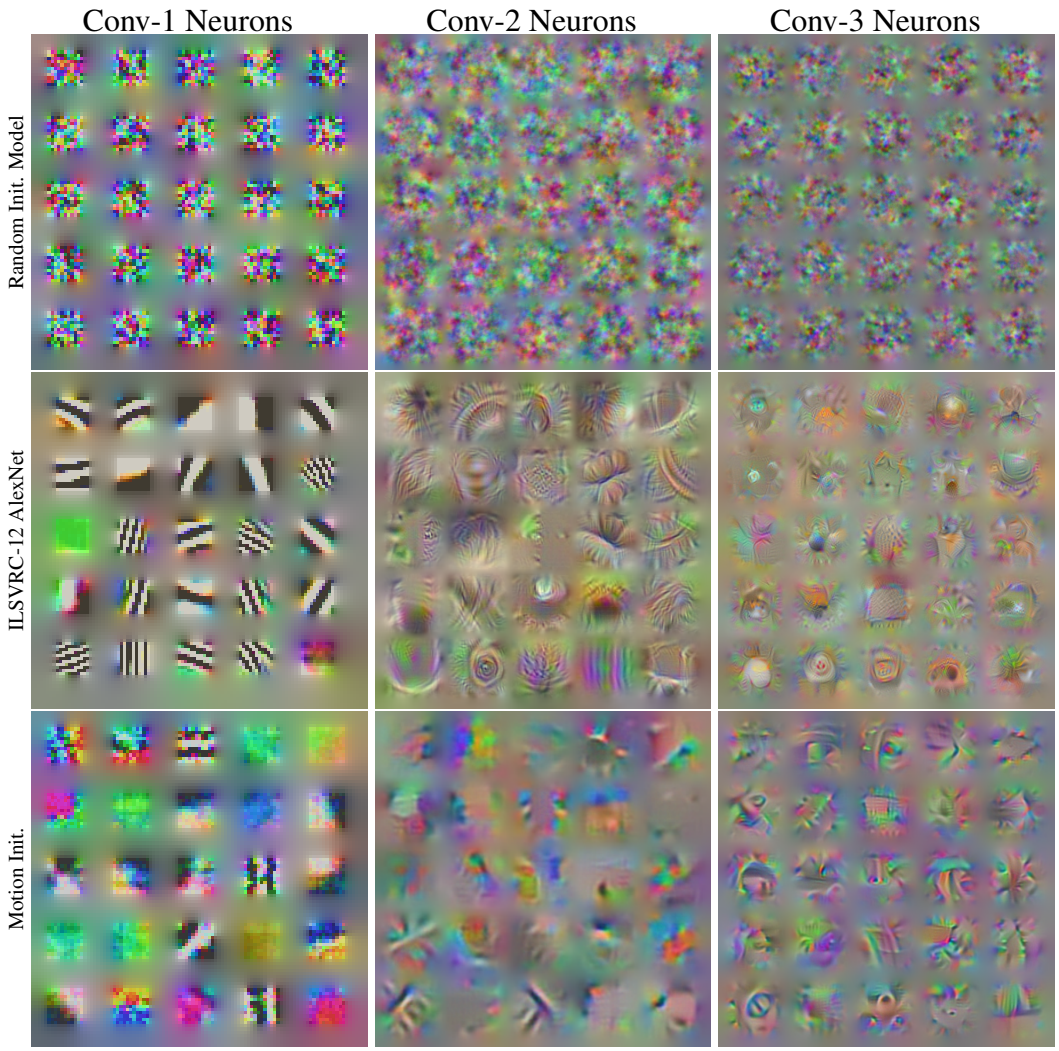


Figure 6.2: Neuron Maximization: Same as fig. 6.1 but for (Conv-1, Conv-2 and Conv-3). Each grid consists of 25 images, one for each neuron.

resembles a human face (See fig. 6.4(left) for a larger image of this neuron's visualization output.). This is not surprising as YFCC100m videos are people centric. This bias is more evident in fig. 6.3 where all visualized neurons in fc6 and fc7 layers show collections of people (See fig. 6.4(right) for a larger image).

In fig. 6.5, we visualize neurons pre-trained using another self-supervised learning method namely Rotation Network (RotNet) [44]. Comparing these visualizations with those from our motion pre-trained model we see richer features in RotNet. This gives some intuition for their superior performance on various quantitative benchmarks. However, we also note that several of these features are horizontal. See for instance, the animal face

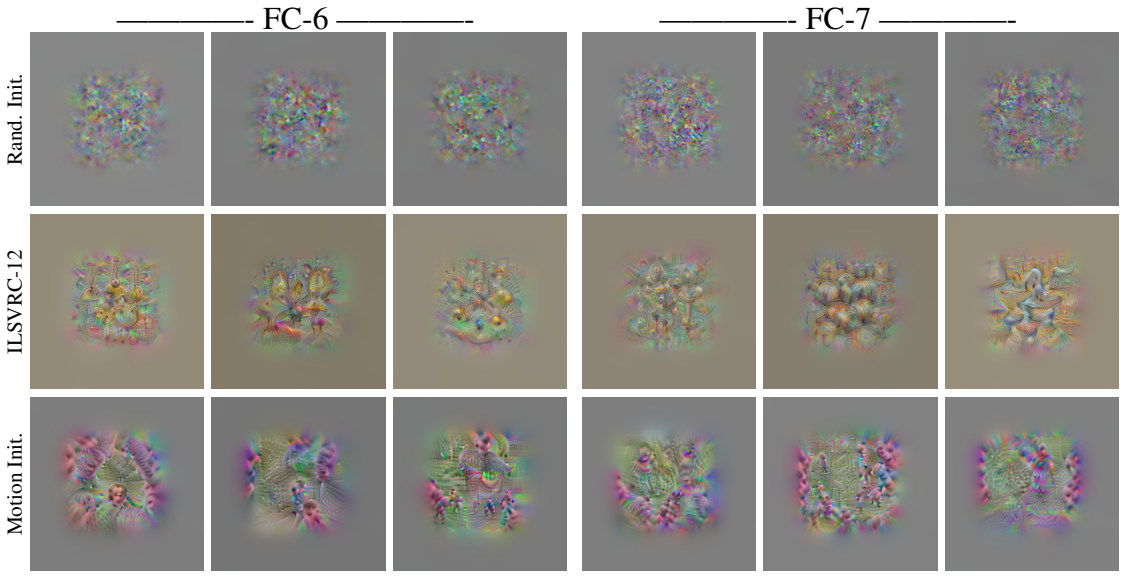


Figure 6.3: Neuron Maximization: Fully Connected Layers (FC-6, FC-7). Each row visualizes the first three neurons in the corresponding fully connected layer.

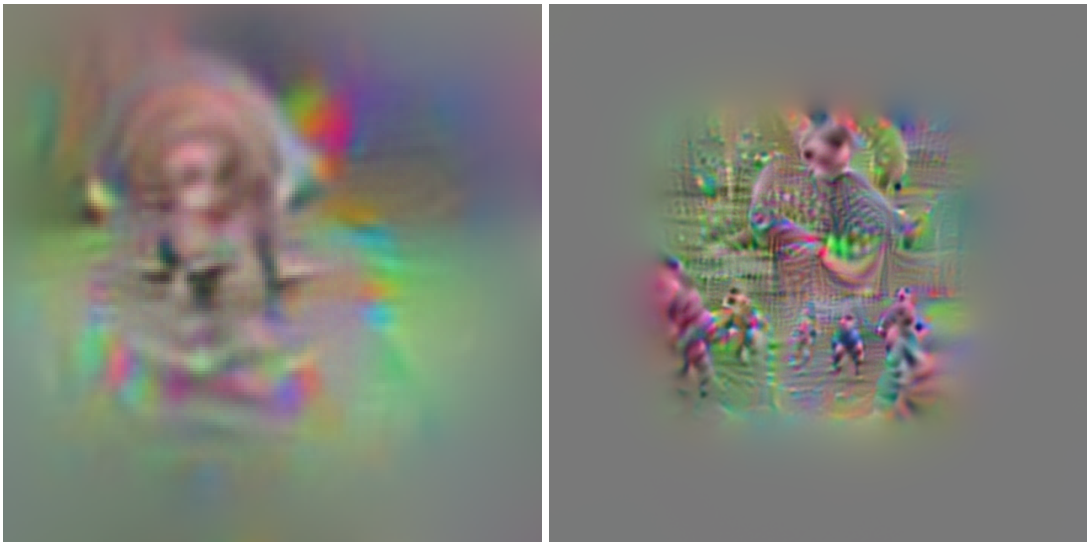


Figure 6.4: Neuron maximization for the CPFS-CE motion pretrained model: (Left) Conv5, neuron 19, Face like neuron, (Right) FC6, Neuron 3, People like neuron.

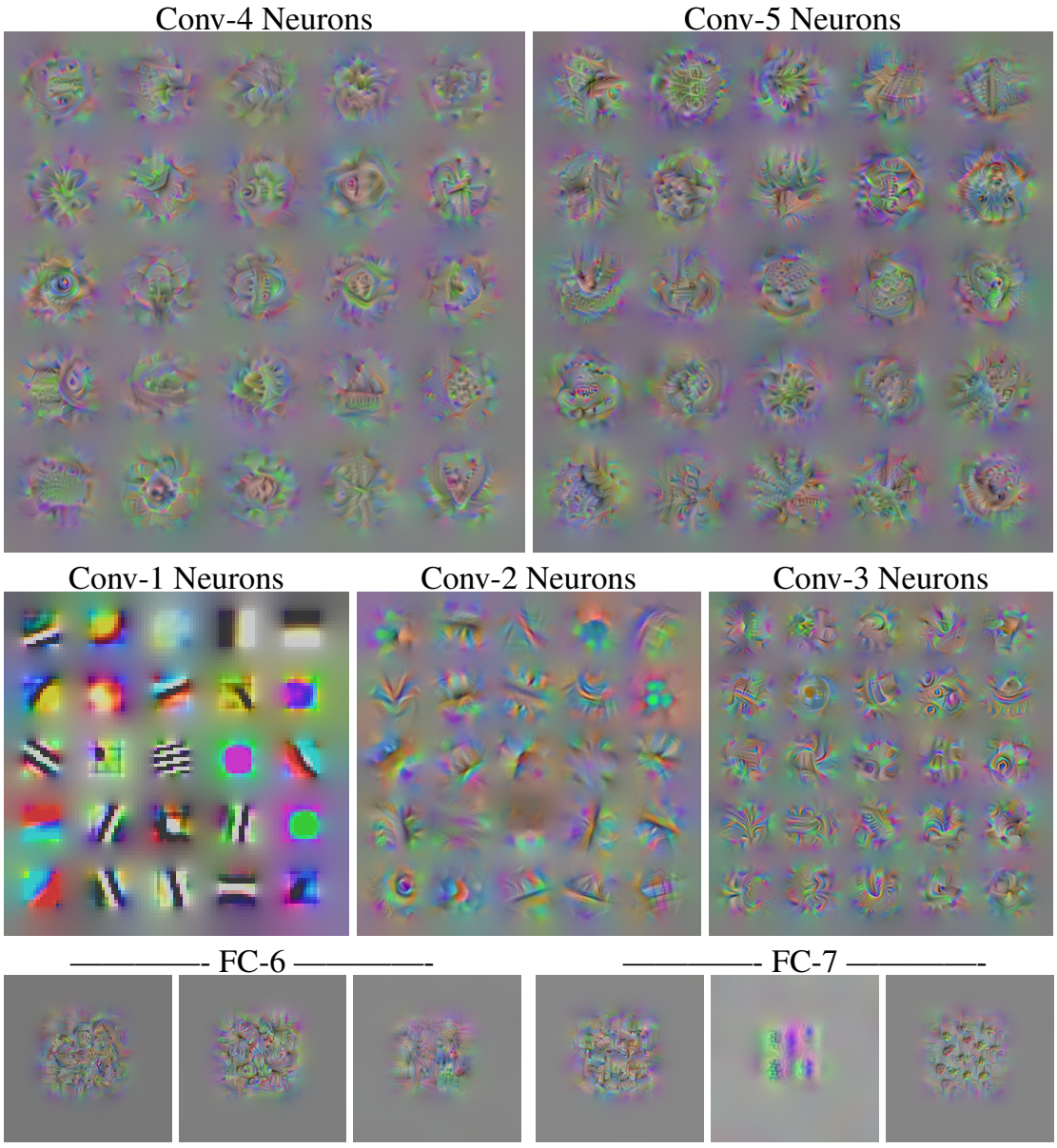


Figure 6.5: RotNet Neuron Maximization: Same as fig. 6.1, fig. 6.2 and fig. 6.3 but for RotNet [44]. Each grid consists of 25 images, one for each neuron.

in row-2-column-4 of the conv4 feature visualization grid. This highlights the effects of training on distorted natural image data. The RotNet pre-training method involves rotating the image by 90/180/270 degrees and predicting this global orientation. Thus the network is trained on images with distorted global orientation leading to unnaturally oriented filters. On the contrary, our motion based pre-training method only loses information along the temporal dimension leaving the image-frame intact.

6.2 Inversion

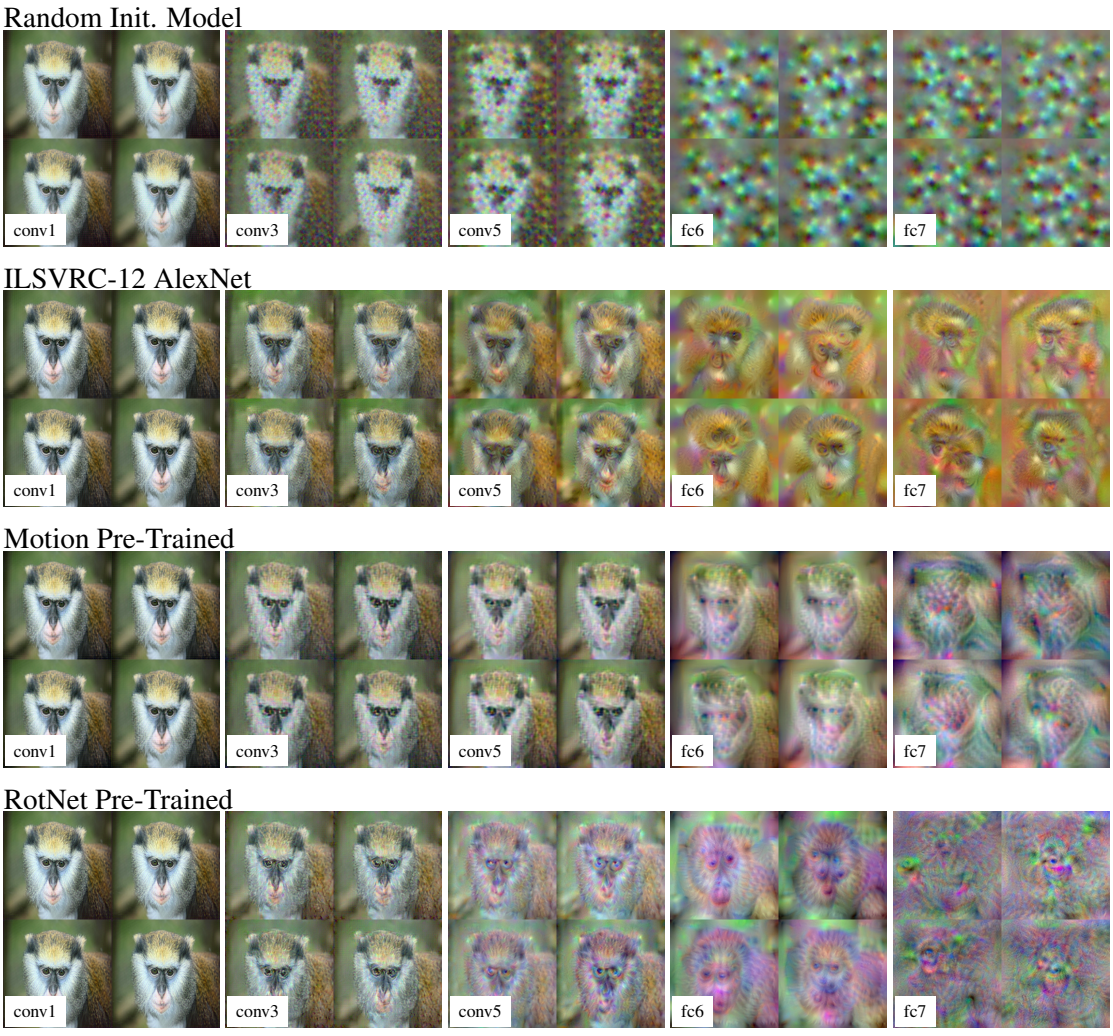


Figure 6.6: Feature Inversion: Multiple inversions of convolutional layers in all four models for features extracted from the "Monkey" image in fig. 5.1. For each layer-model pair, four pre-images are computed and shown in a 2×2 grid.

While neuron maximization visualizes individual neurons, inversion is useful to know

what is captured by a representation as a whole. Figure 6.6 inverts several layers of our motion pre-trained model. These are shown alongside inversions of the same layers for an ImageNet pre-trained AlexNet, a randomly initialized network and another self-supervised pre-trained model (RotNet). It is clear, as was in chapter 5, that going deeper results in a loss of detail. In other words, the network chooses what constitutes irrelevant detail and discards it. The inverted result is then missing this part of information content otherwise present in the image. In the case of a randomly initialized network, it is not selective and visualizations become blurry for deeper layers. On the other hand, both the ILSVRC-12 pre-trained AlexNet and our motion pre-trained model are selective of salient regions such as eyes and outer edges of the object. In fact, object parts are retained in the visualizations of fully connected layers of the ILSVRC-12 pre-trained AlexNet. This is missing for our motion pre-trained network suggesting that the fully connected layers can be improved. RotNet appears to emphasize eyes in its fully connected layers.

6.3 Caricaturization

We observed in fig. 5.19 and fig. 5.20 that maximizing the activity of all active neurons in a neural feature map resulted in the repetition of salient structures in the scene. Here we repeat this experiment for the three networks under consideration in this chapter, in fig. 6.7. We observe that this repetition of object parts is (a) lower in AlexNet when compared to the VGG-M and VGG-VD-16 results discussed in chapter 5, (b) not present in a randomly initialized AlexNet nor in our motion pre-trained model. Instead the latter emphasizes lower level structures highlighting the semantic gap in our pre-training strategy. This gap is particularly evident in the fully connected layers - FC-6 and FC-7.

6.4 Network Dissection

Complementary to the above visualizations, we quantify the number of interpretable neurons in our model by applying the Network Dissection technique of [7]. This method evaluates individual neurons as detectors for several diverse concepts ranging from colors and textures to objects and scenes. A neuron is deemed interpretable if it is

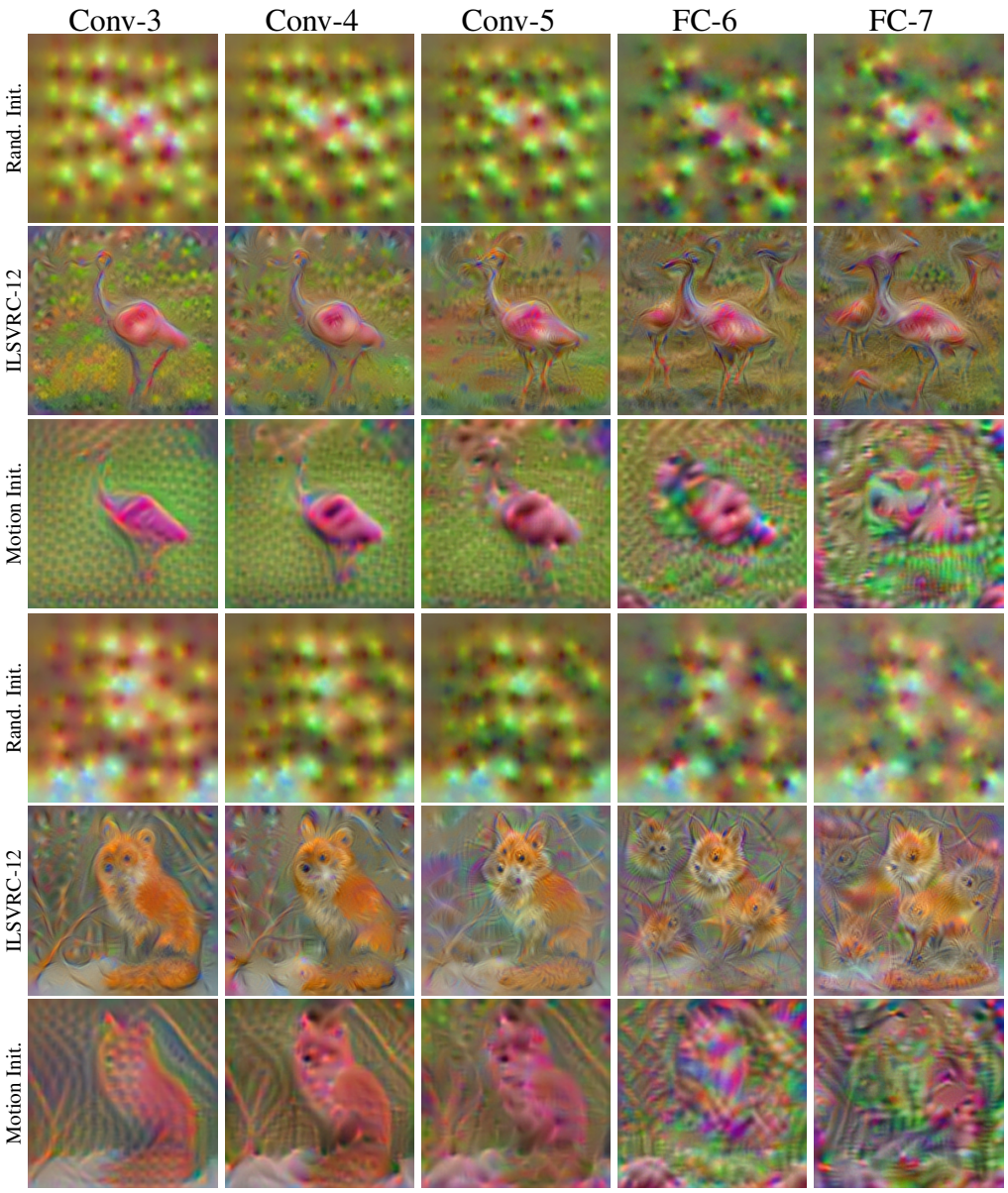
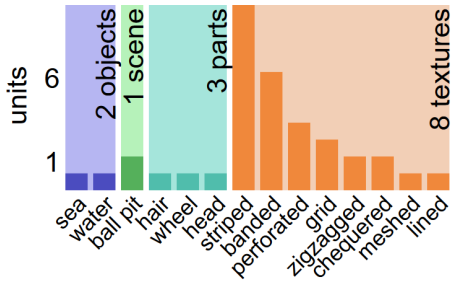
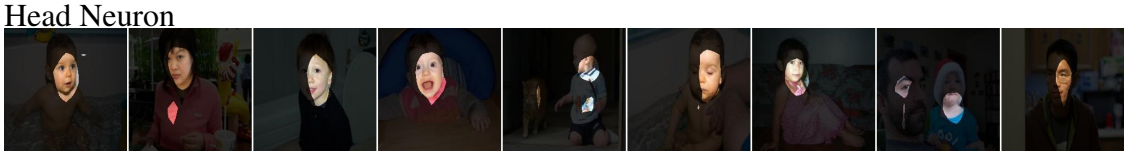


Figure 6.7: Caricaturization for "Flamingo" and "Red-fox" images from fig. 5.1 for three networks: "Rand. Init" - randomly initialized network, "ILSVRC-12" - ILSVRC-12 pre-trained AlexNet, "Motion Init." - Motion based pretrained model from chapter 3.

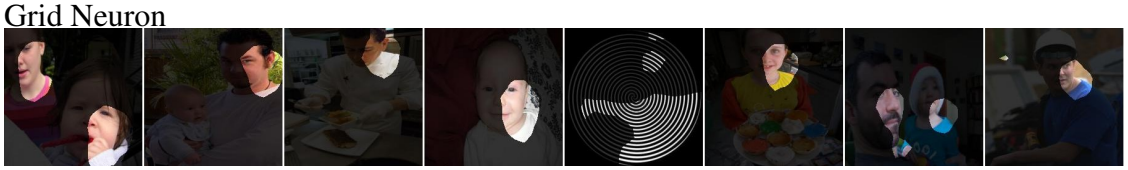
able to detect one of these concepts with a mean intersection over union above 0.04. The authors observed that self-supervised models have fewer interpretable units. This was later attributed to a more distributed representation by [38]. The extent to which such a distributed interpretable representation exists in our CPFS-CE model has already been quantified to some extent in the ImageNet linear probing experiments (Chapter 3, table 3.2). Here we present a summary of *individual* interpretable neurons in the conv5



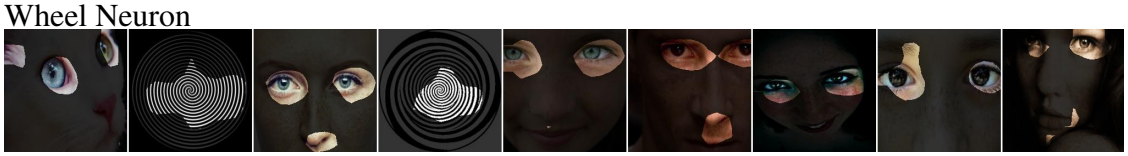
Head Neuron



Grid Neuron

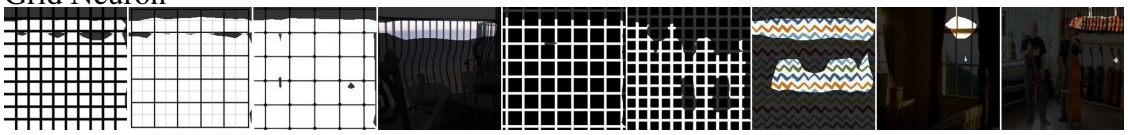


Wheel Neuron



Banded Neuron

Grid Neuron



Striped Neuron

Neuron 19 - Leg detector with $IoU = 0.01$

Figure 6.8: Network Dissection: We quantify the number of individual interpretable neurons, in the conv-5 layer of our CPFS-CE model, using [7]. The histogram at the top enumerates all concepts recognized by individual neurons in the network. We notice that three of the concept detectors - head, grid, wheel - may also be detecting - faces, face patches, eyes - respectively.

layer of our CPFS-CE model in fig. 6.8. We see that most of the concepts are low level textures. It is interesting to again note the bias towards faces as three of the concepts - 'head', 'grid', 'wheel' - are potentially face related. Also note that many neurons are dedicated to textures with repeating geometry such as 'striped', 'banded', 'grid', 'zigzagged'. This observation concurs with the grid pattern like pre-images obtained by neuron maximization of conv4 and conv5 neurons in fig. 6.1. We hypothesize that such patterns help estimate the neuron's distance from the image boundary. Such estimates could help the network solve the non-convolutional proxy task of grouping pixels corresponding to individual object instances. Contrasting the histogram in fig. 6.8 with Figure 12 in [163] we verify a semantic gap between our CPFS-CE model and the ILSVRC-12 pre-trained one.

The 'face' neuron in conv5 is identified as a 'leg' detector ($IoU = 0.01$) by Network Dissection. We believe this is because Network dissection tests whether a given neuron is a detector for one of several concepts. This requires that neuron to 'cover' most instances of that concept. Neuron maximization, however, does not demand such coverage. Also a neuron can detect more than one concept. Network Dissection picks the concept that a neuron detects best and ignores the remaining.

6.5 Summary

We presented visualizations of our self-supervised CPFS-CE model. This constitutes a case study of how a new model may be visualized using our methods. While these visualizations may not convey training specific information by themselves, contrasting them against a randomly initialized model and the best performing ILSVRC-12 pre-trained model gave useful insights. We were able to identify room for improvement and get an intuition for the model's current performance. The key positive aspect of our CPFS-CE model is that it is sensitive to structured patterns rather than noise clouds. The negative aspect is that these structured patterns are not visually resembling semantic entities except for people and faces. Furthermore a combination of these neurons does not capture semantic entities as highlighted by the caricaturization results. This semantic gap was quantitatively verified by using Network Dissection.

We were interested in further visualizing the CPFS-CE model using gradient based methods [121, 126, 156], but as we will see in the next chapter, their characterization of deep layers is confounded by intermediate activations.

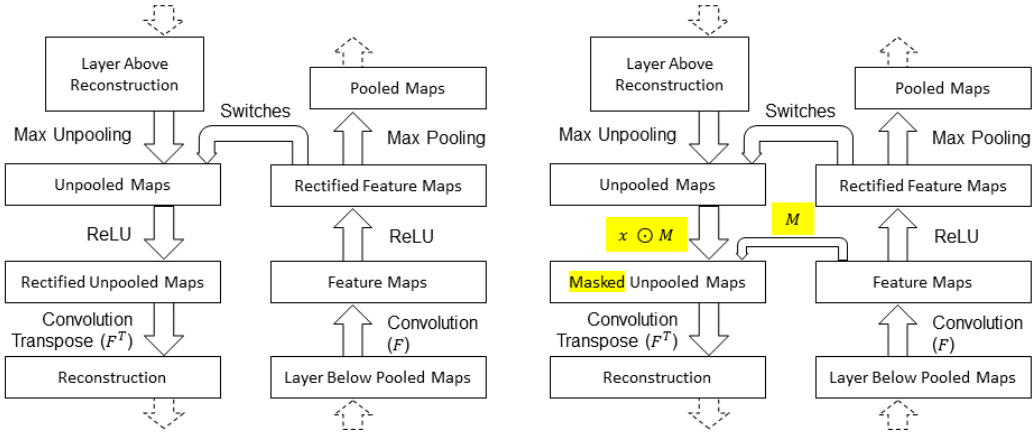
7 Salient deconvolutional networks

This chapter presents an analysis of three recent CNN visualization methods based on network backpropagation namely, Network Saliency [121], Guided Backprop [126] and Deconvolutional Networks (DeConvNet) [156]. We also contribute a weakly supervised foreground segmentation pipeline using these visualization methods as seeds.

7.1 Introduction

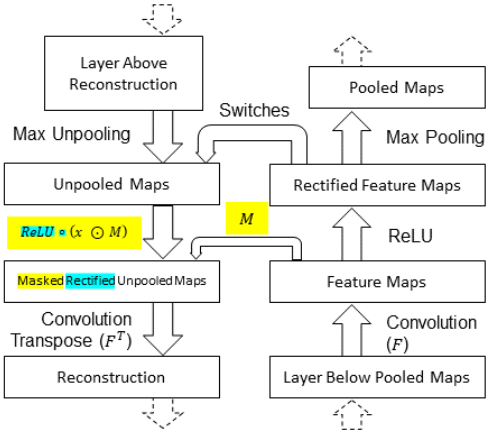
Despite the success of modern Convolutional Neural Networks (CNNs), there is a limited understanding of *how* these complex models achieve their performance. Methods such as DeConvNets have been proposed to *visualize* image patterns that strongly activate any given neuron in a CNN [156] and therefore shed some light on the CNN structure. However, the DeConvNet construction is partially heuristic and so are the corresponding visualizations. Simonyan *et al.* [121] noted similarities with their Network Saliency method which explains DeConvNets, but this interpretation is incomplete.

This chapter carries a novel and systematic analysis of DeConvNets and closely related visualization methods - Network Saliency and Guided Backprop. We first provide a brief description of all three methods before detailing our contributions. Figure 7.1 contains a schematic for each of them emphasizing their differences by highlighting them in yellow and blue.



(a) DeConvNet

(b) Network Saliency



(c) Guided Backprop

Figure 7.1: Schematics illustrating the three methods analyzed in this chapter. These are based on the schematic provided in [156]. In each case, the feed-forward CNN is to the right and the corresponding reversed CNN is to its left. Images/lower layer activations are input to the feed-forward CNN. Information may be copied from the forward CNN to the reversed CNN using lateral connections. Differences between Network Saliency and DeConvNet are highlighted in yellow. Additional differences in Guided Backprop are highlighted in blue.

DeConvNet The DeConvNet construction fig. 7.1a is an architecture reversal strategy contributed by Zeiler and Fergus [156]. Given a feed-forward CNN that maps images to neuron activations, their method constructs a reversed CNN mapping neuron activations back to an image. The reversed CNN is constructed heuristically

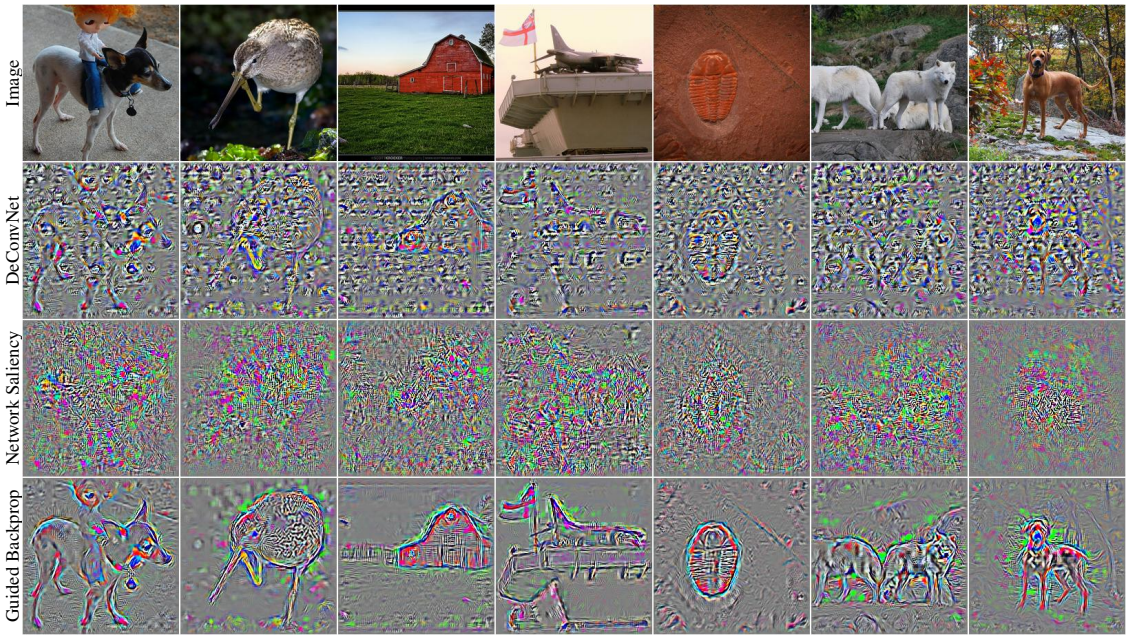


Figure 7.2: From top row to bottom: Original image, DeConvNet, Network Saliency and Guided backprop visualizations from the fc8 layer of AlexNet (just before the softmax operation). The maximally active neuron is visualized in each case. Guided Backprop results in crisper visualizations. It suppresses the background while preserving edge information. Best viewed on screen.

by following three steps: (a) Replace every convolution layer with a convolution transpose layer using the same weights and biases, (b) replace every ReLU (Rectified Linear Unit) layer with a ReLU in the backward direction, and (c) replace every max-pool layer with an "unpool" layer. The unpool layer is discussed mathematically in section 7.3.1. The DeConvNet construction is used to visualize individual neurons by passing a one-hot neuron activation map as input to the reversed CNN. The output image is then assumed to visualize the selected neuron.

Network Saliency The Network saliency algorithm estimates salient regions in an image with respect to a chosen neuron. For this purpose, it defines saliency as the gradient of that neuron's activation with respect to the input image. As noted by Simonyan *et al.* [121] and highlighted in fig. 7.1b, this is identical to the DeConvNet construction except for a difference in the reversal of ReLU layers. M in the figure is the rectification mask, that is a binary tensor identifying positive neurons in the convolutional output. See section 7.3.1 for a formal definition.

Guided Backprop A combination of DeConvNets and Network Saliency yields guided backprop. In this method, the reversed ReLU layer first superimposes the rectification mask M (as in Network Saliency) and next applies $ReLU$ in the backward direction (as in DeConvNets). It can be regarded as an improved version of DeConvNet that is selective of salient regions. See the schematic in fig. 7.1c.

Our first contribution is to extend DeConvNet to a general method for architecture reversal and visualization. In this construction, the reversed layers use information extracted by the forward network, which we call *bottleneck information* (section 7.3). We show that back-propagation is a special case of this construction which is equivalent to the Network Saliency method of Simonyan *et al.* (section 7.3.1). We note that the *only* difference between DeConvNet and Network Saliency is a seemingly innocuous change in the reversal of Rectified Linear Units (ReLU; section 7.3.2). However, this change has a *very significant* effect on the results: the Network Saliency response is well localized but lacks structure, whereas the DeConvNet response accurately reproduces the image boundaries and object shapes, but is less localized (fig. 7.2). A combination of the two methods yields Guided Backprop [126]. We show that it simultaneously obtains structure and localization.

We then move to the important question of whether deconvolutional architectures are useful for visualizing neurons. Our answer is partially negative, as we find that the output of reversed architectures is mainly determined by the bottleneck information rather than by which neuron is selected for visualization (section 7.4.3). In the case of Network Saliency and Guided Backprop, we confirm that the output is selective of any recognizable foreground object in the image, but the class of the selected object cannot be specified by manipulating class-specific neurons.

Having established the dominance of bottleneck information, we draw an analogy between that and phase information in the Fourier transform (section 7.4.4) and discuss the importance of polarity information in reversed architectures.

Finally, we quantitatively test the ability of Network Saliency and Guided Backprop to identify generic foreground objects in images (section 7.4.6). Combined with GrabCut [51], we achieve near state-of-the-art segmentation results on the ImageNet

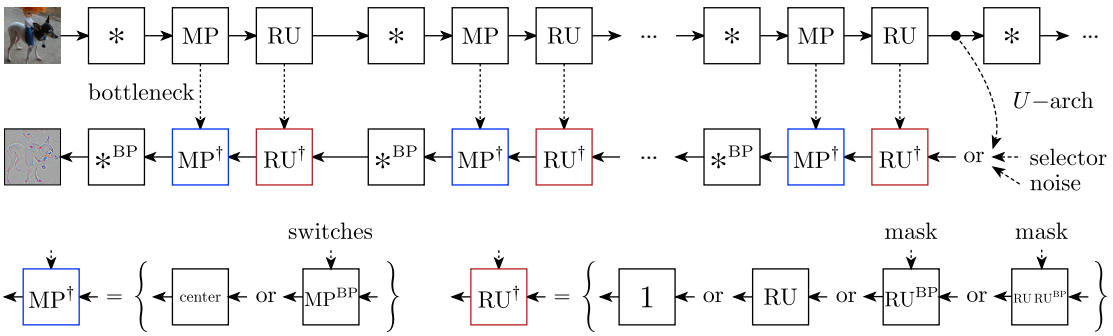


Figure 7.3: The top row shows a typical CNN obtained by repeating short chains of convolution ($*$), max pooling (MP) and ReLU (RU) operators. The middle row shows a generic "deconvolution" architecture, in which information flows backward to the image using the convolution transpose $*^{\mathrm{BP}}$ operator. Different variants are obtained by: (i) choosing a different input (U-architecture, feature selector, or noise), (ii) choosing a variant of backward ReLU $\mathrm{RU}^{\dagger}$ (1, ReLU, ReLU backpropagation, or hybrid), and (iii) choosing a variant of backward max pooling $\mathrm{MP}^{\dagger}$ (unpool to centre or MP backpropagation). This schema generalizes DeConvNets [156].

segmentation task of [50], while using off-the-shelf CNNs pretrained from a *largely disjoint subset* of ImageNet and with only image-level supervision.

7.2 Related Work

DeConvNets were originally proposed as a method for unsupervised feature learning [157, 158] and later applied to visualization [156]. There are several CNN visualizations alternative to DeCovnNets including those discussed in chapter 5. Recently, DeConvNets have also been proposed as a tool for semantic image segmentation; for example,[97, 57] interpolate and refine the output of a fully-convolutional network [85] using a deconvolutional architecture. In this thesis, inspired by [121], we apply reversed architectures for foreground object segmentation, although as a by-product of visualization and in a weakly-supervised setting rather than as a specialized segmentation method.

7.3 Family of Deconvolutional Architectures

Given an image $\mathbf{x} \in \mathcal{X}$, a deep CNN extracts a feature vector or representation

$$\phi : \mathbf{x} \mapsto \phi_L \circ \cdots \circ \phi_1(\mathbf{x}) \tag{7.1}$$

using a sequence of L linear and non-linear layers ϕ_i (fig. 7.3.top). Typical layers include convolution, ReLU, max pooling, and local response normalization.

The goal is to associate with ϕ a corresponding architecture $\phi^\dagger$ that reverses in some sense the computations and produces an image as output. While such reversed architectures have several uses, here we focus on the problem of *visualizing* deep networks: by looking at the images produced by $\phi^\dagger$, we hope to gain some insights about the forward network ϕ. This method was popularized by the work of Zeiler and Fergus in [156], where a particular construction called DeConvNet was shown to produce surprisingly crisp renderings of neural activations. However, given the heuristic nature of some choices in DeConvNet, it is difficult to precisely characterize the meaning of these results.

In order to explore this point, we consider here a generalization of the DeConvNet construction. To this end, each layer ϕ_i is associated with a corresponding layer $\phi_i^\dagger$ that reverses input $\mathbf{x}$ and output $\mathbf{y}$ (fig. 7.3.middle row). We also allow the reverse layer to be influenced by auxiliary information $\mathbf{r}$ computed by the forward layer. For instance, in DeConvNet, the reverse max pooling layer requires the "setting of the pooling switches" computed in the forward pass. Thus a layer ϕ_i and its reverse $\phi_i^\dagger$ are maps:

$$\text{forward } \phi_i : \mathbf{x} \mapsto (\mathbf{y}, \mathbf{r}), \qquad \text{reversed } \phi_i^\dagger : (\hat{\mathbf{y}}, \mathbf{r}) \mapsto \hat{\mathbf{x}}. \tag{7.2}$$

The $\hat{\cdot}$ symbol emphasizes that, in the backward direction, the tensors $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ have the same shape as $\mathbf{x}$ and $\mathbf{y}$ in the forward pass, but different values.

Since the auxiliary information $\mathbf{r}$ is a function of the input $\mathbf{r} = \pi(\mathbf{x})$, one can always let $\mathbf{r} = \mathbf{x}$ without loss of generality; however, the interesting case is when the auxiliary information is limited and $\mathbf{r}$ is an *information bottleneck*. For example, the pooling switches $\mathbf{r}$ in DeConvNet contain much less information than the input data $\mathbf{x}$. In fig. 7.3 these bottlenecks are denoted by dotted arrows.

The question then is how can we build reverse layers $\phi^\dagger$? Next, we show that back-propagation provides a general construction for reverse layers, which in some cases corresponds to the choice in DeConvNet.

7.3.1 Network Saliency as a Deconvolutional Architecture

The *Network Saliency* method of Simonyan *et al.* [121] characterizes which pixels of an image are most responsible for the output of a CNN. Given an image $\mathbf{x}_0$ and a network $\phi(\mathbf{x}_0)$, saliency is computed as the derivative of the (projected) CNN output $S(\phi, \mathbf{x}_0, \mathbf{p})$ with respect to the image:

$$S(\phi, \mathbf{x}_0, \mathbf{p}) = \frac{\partial}{\partial \mathbf{x}} \langle \mathbf{p}, \phi(\mathbf{x}) \rangle \Big|_{\mathbf{x}=\mathbf{x}_0}. \tag{7.3}$$

Since the CNN output $\phi(\mathbf{x})$ is in general a vector or tensor, the latter is transformed into a scalar by linear projection onto a constant tensor $\mathbf{p}$ before the derivative is computed. In practice, $\mathbf{p}$ is usually a one-hot tensor that selects an individual neuron in the output. In this case, the value of a pixel in the saliency map $S(\phi, \mathbf{x}_0, \mathbf{p})$ answers the question: "how much would the neuron response $\langle \mathbf{p}, \phi(\mathbf{x}_0) \rangle$ change by slightly perturbing the value of that pixel in the image $\mathbf{x}_0$?"

Given network ϕ with L layers, saliency is computed from eq. (7.1) and eq. (7.3) using the chain rule:

$$\operatorname{vec} S(\phi, \mathbf{x}_0, \mathbf{p}) = \operatorname{vec} \mathbf{p}^\top \times \frac{\partial \operatorname{vec} \phi_L}{\partial \operatorname{vec} \mathbf{x}_{L-1}^\top} \times \cdots \times \frac{\partial \operatorname{vec} \phi_1}{\partial \operatorname{vec} \mathbf{x}_0^\top}. \tag{7.4}$$

where $\mathbf{x}_l$ are activations of the l^{th} layer. Here the vec operator stacks tensors into vectors and allows us to use a simple matrix notation for the derivatives.

The *Back Propagation* (BP) algorithm [116] is the same as computing the products eq. (7.4) from left to right; this reduces to a chain of derivatives in the form of eq. (7.3), one for each layer, where $\mathbf{p}$ is replaced with the derivative $\hat{\mathbf{y}}$ obtained from the layer above. In this manner, BP provides a general way to define a reverse of any layer ϕ_i:

$$\phi_i : \mathbf{x} \mapsto \mathbf{y} \quad \text{BP-reversed becomes} \quad \phi_i^{\mathrm{BP}} : (\mathbf{x}, \hat{\mathbf{y}}) \mapsto \frac{\partial}{\partial \mathbf{x}} \langle \hat{\mathbf{y}}, \phi_i(\mathbf{x}) \rangle. \tag{7.5}$$

We denote the BP-reversed of a layer with the symbol ϕ_i^{BP}. Any CNN toolbox can compute BP-reversed for any layer as it contains code for back-propagation. Note also that the BP-reversed layer is *a linear map* in the argument $\hat{\mathbf{y}}$, even if the forward layer is non-linear in $\mathbf{x}$. In this manner, one can compute backpropagation, and therefore the

saliency map $S(\phi, \mathbf{x}_0, \mathbf{p})$ of [121]. This computation in principle uses a "deconvolutional" architecture of the type of fig. 7.3, where layers are reversed using the BP equation (7.5).

The BP-reversed layer ϕ_i^{BP} takes as input both $\mathbf{x}$ and $\hat{\mathbf{y}}$, whereas from our discussion above we would like to replace $\mathbf{x}$ with a bottleneck $\mathbf{r}$. Formally we rewrite the BP-reversed layer $\phi_i^{\mathrm{BP}}(\mathbf{x}, \hat{\mathbf{y}})$ as $\phi_i^{\dagger}(\hat{\mathbf{y}}, \mathbf{r})$ where $\mathbf{r} = \pi(\mathbf{x})$ projects the data $\mathbf{x}$ onto the smallest possible bottleneck. Note that this does not change the meaning of a BP-reversed layer, but it does characterize how much auxiliary information it requires. The latter is easy to find in an abstract sense,[1] but it is much more instructive to derive it for concrete layer types, which we do below for common layers.

Affine layers. A *fully connected layer* ϕ_{fc} simply multiplies the data $\mathbf{x}$ by a matrix A and adds a bias b. Given that the data $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ is a 3D tensor of height H and width W and C feature channels, we use the vec operator to write this in terms of matrices[2] as $\phi_{\mathrm{fc}} : \mathrm{vec}\,\mathbf{y} = A\,\mathrm{vec}\,\mathbf{x} + b$. *Linear convolution* ϕ_* can conveniently be defined in the same way, by replacing matrix A with a matrix $\rho(F)$ constructed by "sliding" a bank of filters F, giving $\phi_* : \mathrm{vec}\,\mathbf{y} = \rho(F)\,\mathrm{vec}\,\mathbf{x} + b$. Using eq. (7.5), the BP-reversed layers are obtained by transposing the respective matrices:

$$\phi_{\mathrm{fc}}^{\mathrm{BP}} : \mathrm{vec}\,\hat{\mathbf{x}} = A^\top\,\mathrm{vec}\,\hat{\mathbf{y}}, \qquad \phi_*^{\mathrm{BP}} : \mathrm{vec}\,\hat{\mathbf{x}} = \rho(F)^\top\,\mathrm{vec}\,\hat{\mathbf{y}}. \tag{7.6}$$

The layer ϕ_*^{BP} is often called *deconvolution*[3] and gives the name to DeConvNets.

Note that the computation of these layers does not require any information from the forward pass, so the *bottleneck* $\mathbf{r}$ *is empty*. This is due to linearity and explains why in fig. 7.3 there are no dashed arrows connecting convolutional layers.

Rectified linear Unit (ReLU or RU). ReLU and its BP-reversed layer are given by

$$\phi_{\mathrm{RU}}(\mathbf{x}) = \max\{\mathbf{x}, 0\}, \qquad \phi_{\mathrm{RU}}^{\mathrm{BP}}(\mathbf{x}, \hat{\mathbf{y}}) = \phi_{\mathrm{RU}}^{\dagger}(\hat{\mathbf{y}}, \mathbf{r}) = \hat{\mathbf{y}} \odot \mathbf{r}, \quad \mathbf{r} = [\mathbf{x} > 0], \tag{7.7}$$

[1] Let $\mathbf{x}' \sim \mathbf{x}''$ be equivalent whenever functions $\phi_i^{\mathrm{BP}}(\mathbf{x}', \cdot) = \phi_i^{\mathrm{BP}}(\mathbf{x}'', \cdot)$ are the same. It is easy to check that this defines an equivalence relation. Then the smallest possible bottleneck $\pi : \mathbf{x} \mapsto \mathbf{r} \in \mathcal{X}/\sim$ projects $\mathbf{x}$ into its equivalence class.

[2] This is slightly more general than usual as it specifies a different bias for each output dimension instead for each output feature channel.

[3] The name "deconvolution" is technically incorrect here. The operation itself is a "Convolution Transpose".

where $\max$ (maximum) operation is computed element-wise, $\odot$ is the element-wise product, and $[\mathbf{x} > 0]$ is the rectification mask (binary tensor) with a 1 for every positive element of $\mathbf{x}$ and 0 otherwise.

Hence the *bottleneck information for ReLU is the rectification mask* $M = \mathbf{r} = [\mathbf{x} > 0]$. Note that $\phi^{\mathrm{BP}}_{\mathrm{RU}}(\mathbf{x}, \hat{\mathbf{y}})$ is not the reversal used by DeConvNets [156, 121] and this choice changes the output significantly.

Max Pooling (MP). Let x_{uc} be the element of tensor $\mathbf{x}$ at spatial location $u \in \Omega$ and feature channel c. MP is obtained by computing the maximum of x_{vc} over a small spatial neighbourhood, $v \in N(u) \subset \Omega$, corresponding to u:

$$[\phi_{\mathrm{MP}}(\mathbf{x})]_{uc} = \max_{v \in N(u)} x_{vc} = x_{s(u|c,\mathbf{x}),c} \quad \text{where} \quad s(u|c,\mathbf{x}) = \operatorname*{argmax}_{v \in N(u)} x_{vc}. \tag{7.8}$$

Here $s(u|\mathbf{x}, c)$ tells which element $x_{s(u|\mathbf{x},c),c}$ of the input is associated by maximum to each element y_{uc} of the output and is informally called a *setting of the pooling switches*. A short derivation from eq. (7.5) shows that the BP-reversed is given by

$$[\phi^{\mathrm{BP}}_{\mathrm{MP}}(\mathbf{x}, \hat{\mathbf{y}})]_{vc} = [\phi^{\dagger}_{\mathrm{MP}}(\hat{\mathbf{y}}, \mathbf{r})]_{vc} = \sum_{u \in s^{-1}(v|c,\mathbf{x})} \hat{y}_{uc}, \qquad \mathbf{r} = s(\cdot|\cdot, \mathbf{x}). \tag{7.9}$$

Hence the *bottleneck information for MP is the setting of the pooling switches*.

7.3.2 Deconvolutional Architectures

BP-reversal is only one way of defining reversed layers in a deconvolutional architecture. Here, we examine seven others. These are obtained by choosing between various options for reversing max-pooling and ReLU layers. First for max-pooling layers, we can choose between (a) the BP-reversed $\phi^{\mathrm{BP}}_{\mathrm{MP}}$ layer or (b) simply unpooling to the center of each neighbourhood. The latter has an empty bottleneck. We refer to it as $\phi^{\mathrm{center}}_{\mathrm{MP}}$. Next for ReLU layers, we can choose between: (a) BP-reversed $\phi^{\mathrm{BP}}_{\mathrm{RU}}$ with the rectification mask as bottleneck, (b) identity function 1, with empty bottleneck, (c) ReLU in the backward direction, say ϕ_{RU}, as in DeConvNets, or (d) $\phi_{\mathrm{RU}} \circ \phi^{\mathrm{BP}}_{\mathrm{RU}}$ combining (a) and (c) as in Guided Backprop. Taking all possible combinations between the two sets of choices above, we get eight possible deconvolutional architectures. Example outputs for all eight architectures are shown in fig. 7.4 and fig. 7.5. Note that only three of

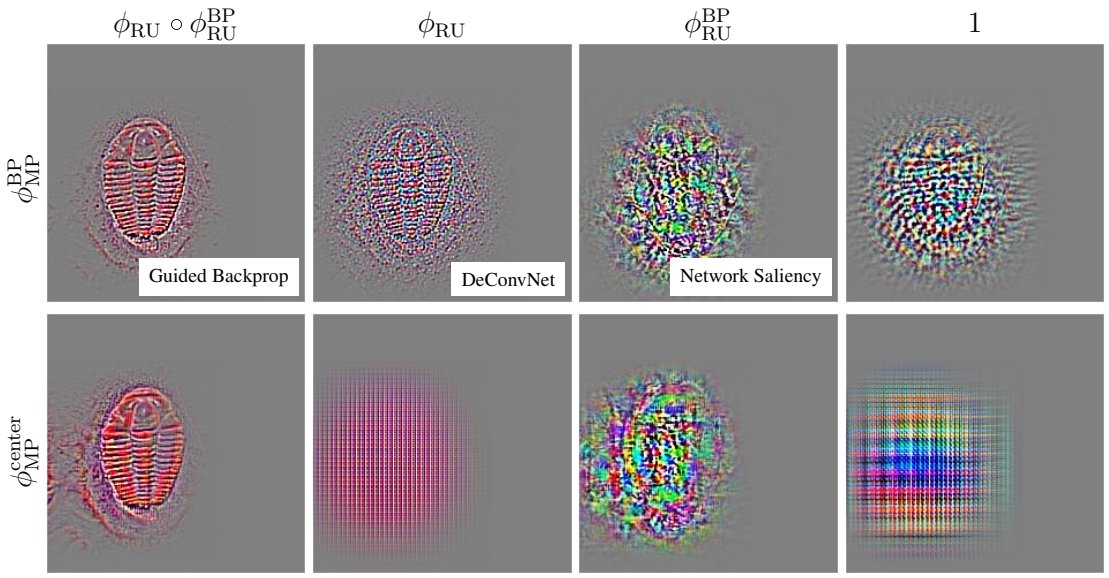


Figure 7.4: Visualizations of VGG-16 using the *trilobite* image of fig. 7.2 (Col. 5) using eight deconvolutional architectures. These architectures are used to visualize the maximally-firing neuron in the *pool5* layer. Full output image is shown (localization is mostly due to the finite support of the neuron). From top to bottom we change $\phi_{\mathrm{MP}}^{\dagger}$ to choose between $\phi_{\mathrm{MP}}^{\mathrm{BP}}$ and $\phi_{\mathrm{MP}}^{\mathrm{center}}$. From left to right we change $\phi_{\mathrm{RU}}^{\dagger}$ to choose between $\phi_{\mathrm{RU}} \circ \phi_{\mathrm{RU}}^{\mathrm{BP}}$, ϕ_{RU}, $\phi_{\mathrm{RU}}^{\mathrm{BP}}$ and 1. Here all methods use the identity as the reverse $\mathrm{LRN}^{\dagger}$.

these, Network Saliency, DeConvNet and Guided Backprop, have been recognized by prior work of which only Network Saliency has an immediate interpretation, in that it is computing the derivative of the forward network.

Affine layers, max pooling, and ReLU cover all the layer types needed to reverse architectures such as VGG-VD, GoogLeNet, Inception and ResNet.[4] AlexNet includes *local response normalization* (LRN) layers, which in DeConvNet are reversed as the identity. As discussed in the section 7.4.5, this has little effect on the results. In the rest of this chapters, DeConvNet and Guided Backprop use identity, while Network Saliency, in keeping with the original saliency method by [121], uses $\phi_{\mathrm{LRN}}^{\mathrm{BP}}$.

7.4 Experiments

Experiments thoroughly investigate the family of deconvolutional architectures identified in section 7.3.2. Section 7.4.1 tests eight possible network architectures and identifies DeConvNet, Network Saliency, and Guided Backprop as interesting cases for further

[4] In all cases we deal with the network only till the layer before the softmax

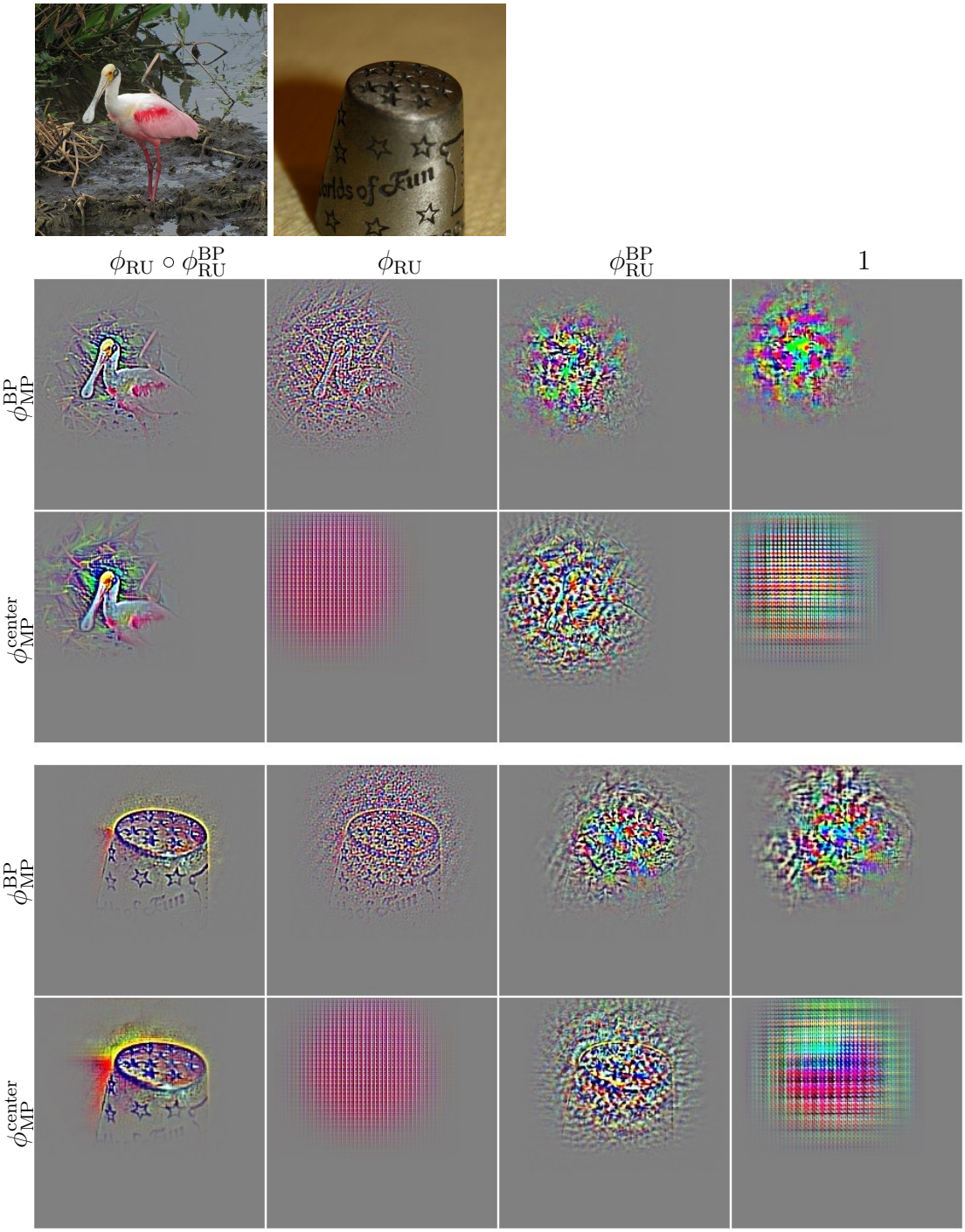


Figure 7.5: Eight Deconvolutional Architectures: Visualizations of VGG-16 pool5 similar to those in fig. 7.4 for two other images.

exploration. Section 7.4.2 compares the architectures in terms of clarity of the generated images. Section 7.4.3 investigates whether visualizations provide useful information about neurons, and section 7.4.4 looks at the effect of the bottleneck information. Finally, section 7.4.6 evaluates these techniques on a practical application: segmentation of foreground objects. Several experiments are shown qualitatively for a few representative images from the top row of fig. 7.2.[5]

7.4.1 Overview of Deconvolutional Architectures

The first experiment compares the eight deconvolutional architectures of section 7.3.2. This is done by "reversing" the computations obtained when a network ϕ is evaluated on an image $\mathbf{x}_0$ (for instance, the "trilobite" image of fig. 7.2 column 5). In this experiment the forward network $\mathbf{y} = \phi(\mathbf{x}_0)$ is VGG-VD-16 [123] truncated at the last max-pooling layer (pool5). The input $\mathbf{p}$ to the reversed network $\phi^\dagger(\mathbf{p})$ is the one-hot tensor selecting the pool5 neuron y_{uc} that is maximally excited by $\mathbf{x}_0$. We vary $\phi^\dagger$ across the eight deconvolutional architectures and compare their outputs qualitatively in fig. 7.4 and fig. 7.5.

We can make the following observations. First, as in [121], Network Saliency computes a fuzzy saliency map. Likewise, matching the results of [156], the result of DeConvNet has structure, in the sense that object edges are recovered.

Second, we compare the left four deconvolutional architectures (columns 1 and 2) to the right ones (columns 3 and 4), which differ by the use of the ReLU units in the backward direction. We note that adding these units is *necessary* in order to recover the image edges. In particular, by modifying Network Saliency in this manner, Guided Backprop produces an image with structure (Compare row1-col1 and row1-col3 of fig. 7.4).

Third, using pooling switches (top row) slightly improves the clarity of the results compared to unpooling to center (bottom row). Even so, we note that the image structure can still be clearly recognized in the bottom-left image, using unpooling to center complemented by the hybrid $\mathrm{RU} \circ \mathrm{RU}^{\mathrm{BP}}$. In fact, this image is arguably crisper than

[5] To improve the clarity of visualization images $\mathbf{x} = \phi^\dagger(\mathbf{p})$ in print, their real valued ranges are remapped using the expression $\sigma(\mathbf{x}/(-\log(99)a))$ where a is the 0.5% quantile in $\operatorname{vec} I$.

the DeConvNet result. This suggests that, perhaps unexpectedly, the ReLU polarity (captured by RU in the backward direction) is more important that the MP switches. It also shows that the ReLU masks (captured by $\mathrm{RU}^{\mathrm{BP}}$) significantly improve the sharpness of the results.

7.4.2 Generated Image Quality

A first striking property of Guided Backprop is the clarity of resulting visualizations compared to the other architectures (*e.g.* fig. 7.2, fig. 7.4, fig. 7.6 fig. 7.8). Sharper visualizations than Network Saliency are expected based on results in [126]. The gap with DeConvNet is, however, unexpected and particularly strong for deep layers (*e.g.* fig. 7.2) and deeper architectures (*e.g.* fig. 7.8). DeConvNet results for AlexNet (fig. 7.6 col. 1-2) appear to be less sharp than the ones shown in [156] (See fig.2 in their paper). This could be due to the fact that they used a custom version of AlexNet, whereas we visualize off-the-shelf versions of AlexNet and VGG-VD. Unfortunately, it was not possible obtain a copy of their custom AlexNet to verify this hypothesis.

7.4.3 Meaning and Selectivity of the Deconvolutional Response

Visualizations obtained using reversed architectures such as DeConvNets are meant to characterize the *selectivity of neurons* by finding which visual patterns cause a neuron to fire strongly. However, we will see here that this interpretation is fragile.

Consider the i-th neuron $[\phi(\mathbf{x})]_i = \langle \mathbf{e}_i, \phi(\mathbf{x}) \rangle$ in the output layer of a (truncated) CNN architecture, where $\mathbf{e}_i$ is an indicator vector. In order to characterize this neuron, Zeiler and Fergus [156] search a large collection of images to find an image $\mathbf{x}^*$ that causes $\phi_i(\mathbf{x}^*)$ to respond strongly. Thus, even before the deconvolutional network $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is applied, the image $\mathbf{x}^*$ is already representative of the neuron. The application of $\phi^\dagger$ then *refines* this information by highlighting which regions in $\mathbf{x}^*$ are most responsible for this activation.

While this sounds simple, there is a subtle complication. Note in fact that the deconvolutional architecture $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is a function both of the neuron indicator $\mathbf{e}_i$ as well as the bottleneck information $\mathbf{r}$ extracted from the forward pass of $\mathbf{x}^*$ through $\phi(\mathbf{x})$. In the deconvolution process, $\mathbf{e}_i$ is a direct specification of the neuron to be visualized. The

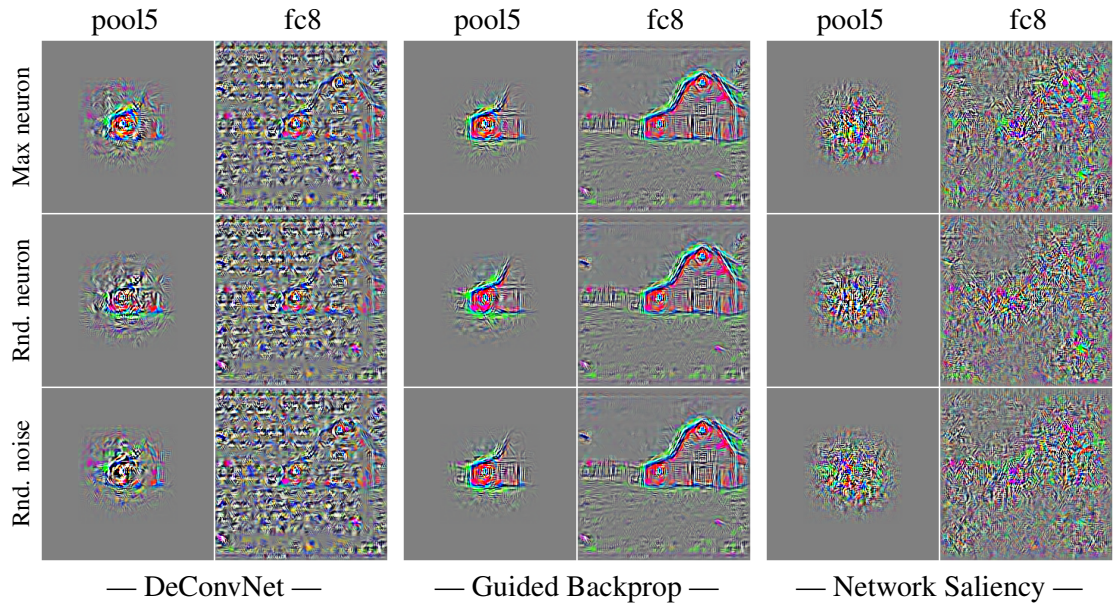


Figure 7.6: *Lack of neuron selectivity.* The bottleneck information $\mathbf{r}$ is fixed to the one computed during the forward pass $\phi(\mathbf{x})$ through AlexNet and the output of $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is computed by choosing $\mathbf{e}$ as: the most active neuron (top row), a second neuron at random (middle), or as a positive random mixture of all neurons (bottom row). Results barely differ, particularly for the deeper layers. See fig. 7.2 for the original house input image $\mathbf{x}$. Best viewed on screen.

other parameter, $\mathbf{r}$, can also be considered as a specification of the same neuron, although a fairly indirect one, because it is extracted from an image $\mathbf{x}^*$ that happens to excite the neuron strongly. Then the question is whether the deconvolutional response can be interpreted as a *direct* characterization of a neuron or not. This is answered next.

Lack of neuron selectivity. If the output of $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is a direct characterization of the i-th neuron, we would expect the generated image to *meaningfully change* as the input $\mathbf{e}_i$ to the deconvolutional network changes.

In fig. 7.6, DeConvNet, Guided Backprop, and Network Saliency are used to visualize the responses of different neurons at the center of the image. In more detail, the reversed function $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is evaluated by keeping $\mathbf{r}$ fixed (as obtained from the forward pass $\phi(\mathbf{x}_0)$) and by replacing $\mathbf{e}$ with either: the indicator vector $\mathbf{e}^*$ of the neuron that has the maximal response, a second random neuron $\mathbf{e}'$ that still generates a non-zero image, and a random non-negative vector $\mathbf{e}$. It can be noted that, particularly in deeper layers, the response changes very little with different choices of $\mathbf{e}$.

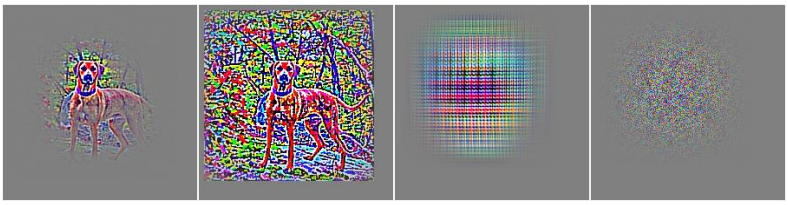


Figure 7.7: *Effect of finite neuron support.* From left to right: Visualization from VGG-16 pool5 using Guided Backprop; the same result after $\mathbf{x}^{0.1}$ renormalization; visualization without any bottleneck information as in fig. 7.4-bottom right; the same visualization without bottleneck information but with randomized filter weights for the $\circledast^{\mathrm{BP}}$ operators. The re-normalization reveals that the true receptive field of pool5 is much larger and that the sides are not discarded but simply weakened in the deconvolution process.

A clear difference between images from different depths (*e.g.* pool5 vs fc8 in fig. 7.6 and fig. 7.8) is the extent of the response, which however corresponds to the neuron support and depends on the architecture and not on the learned network weights or data. This is further confirmed in fig. 7.7 by considering a network with random weights. There, it is also shown that re-normalizing the image intensities reveals the full neuron support, which is only partially suppressed in the visualization, and in a manner which is architecture-dependent rather than weight or data dependent.

We conclude that the outputs of reversed architectures $\phi^{\dagger}(\mathbf{e}, \mathbf{r})$, for deep models ϕ, are mainly dependent on the bottleneck information $\mathbf{r}$ rather than the neuron selector $\mathbf{e}$. Hence, they provide poor direct characterizations of neurons, particularly of deep ones.

Note that methods such as [121, 154], which visualize individual neurons by activation maximization, are not affected by this problem. There are two reasons: first, they start from random noise, such that the bottleneck information $\mathbf{r}$ is *not* primed by a carefully-selected reference image $\mathbf{x}_0$; secondly, they iteratively update the bottleneck information, drifting away from the initial value.

Foreground object selectivity. Network Saliency [121] showed that the deepest class-specific neurons (in fc8) in an architecture such as AlexNet are strongly selective for the foreground objects in an image. However, in the previous section we have shown the apparently contradictory result that this response depends very weakly on the chosen class-specific neuron.

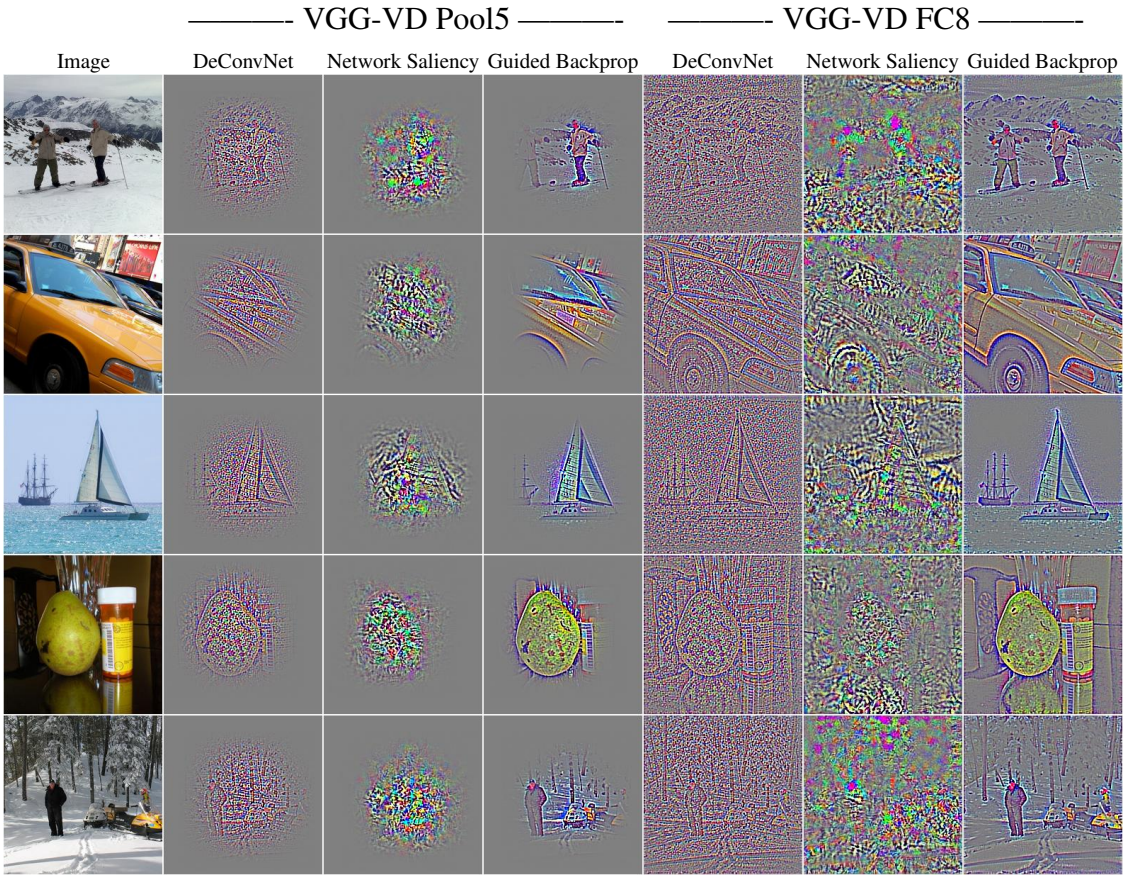


Figure 7.8: *Foreground object selectivity.* This figure compares the response of DeConvNet, Network Saliency, and Guided Backprop by visualizing the most active neuron in Pool5 and FC8 of VGG-VD. Network Saliency and Guided Backprop tend to emphasize foreground objects (see e.g. the faces of people), whereas DeConvNet's response is nearly uniform. Note that the apparent spatial selectivity of Pool5 is due to the finite support of the neuron. Best viewed on screen.

To clarify this point, in fig. 7.8 we observe that Network Saliency and Guided Backprop *are indeed* selective for the foreground object in the image; however, the information *comes mainly from the bottleneck* $\mathbf{r}$ and not from which specific neuron is selected. In other words, Network Saliency and Guided Backprop emphasize whatever foreground object is detected by the network in the forward pass, regardless of which neuron is specified as input to the reversed architecture. The main difference between Network Saliency and Guided Backprop, as observed before, is that the latter produces a much more localized and crisp response.

Compared to Network Saliency and Guided Backprop, DeConvNet fails to produce a clearly selective signal from these very deep neurons, generating a rather uniform response. We conclude that saliency, in the sense of foreground object selectivity, requires

not only the max pooling switches (used by all three architectures), but also the ReLU rectification masks (used only by Network Saliency and Guided Backprop).

Informativeness of bottleneck. In order to characterize the amount of information contained in the bottleneck, we use the method of [26]. They learn the filters of a reversed CNN with the goal of approximately inverting a pre-trained feed-forward CNN. While the inverse network of [26] operates only from the output of the feed-forward model, here we modify it to use different amounts of bottleneck information. In particular, we consider the DeConvNet, Guided Backprop and Network Saliency constructions of reversed networks as CNNs with learnable parameters $\phi^\dagger(\mathbf{y}, \mathbf{r}; W^\dagger)$. These parameters are optimized to minimize the squared L^2 loss on mean-subtracted RGB pixels $\mathbf{x}$.

$$E(\mathbf{r}) = \min_{W^\dagger} ||\phi^\dagger(\phi(\mathbf{x}), \mathbf{r}(\mathbf{x}); W^\dagger) - \mathbf{x}||_2^2 \tag{7.10}$$

The reconstruction error E of these "informed" inverse networks is assumed to illustrate how informative the bottleneck is. The lower the error the more informative the bottleneck. We found that inverting with the knowledge of the ReLU rectification masks and the MP pooling switches has 15% lower L^2 reconstruction error (on validation images) than using pooling switches alone, and 46% lower than using the rectification masks alone. Finally, pooling switches alone have 36% lower L^2 error than using only the rectification masks. Thus ReLU Mask $<$ Pooling Switches $<$ ReLU Mask + Pooling Switches, in increasing order of informativeness.

7.4.4 Dominance of "Phase" Information

If a message emerges from the previous sections, it is that the response of all reversed architectures $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is *dominated by the bottleneck information* $\mathbf{r}$. As seen in section 7.3, the bottleneck information comprises 1) the setting of the pooling switches in the max pool units and 2) the setting of the masks in the ReLU units.

Interestingly, this information does not code for the intensity of the neural activations, but rather for their spatial location and polarity. We argue that this information is somewhat similar to phase information in the Fourier transform and related representations. To explain this analogy, consider the Fourier transform $X(\omega_x, \omega_y) = \mathcal{F}[\mathbf{x}](\omega_x, \omega_y) \in \mathbb{C}$

of image $\mathbf{x}$; a well known result is that if one replaces the modulus of the Fourier transform with a random signal but preserves the phase, then the reconstructed image $\hat{\mathbf{x}} = \mathcal{F}^{-1}\big[|Y(\omega_x, \omega_y)|e^{i\angle X(\omega_x,\omega_y)}\big]$ still contains the structure (edges) of $\mathbf{x}$ and very little of $\mathbf{y}$ is recognizable. In fact the resulting image, an example of which is shown in fig. 7.9, is not dissimilar from the output of DeConvNet and Guided Backprop.

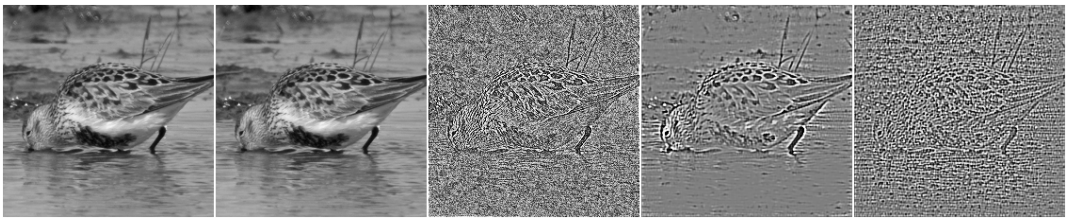


Figure 7.9: *Analogy with phase information in Fourier Transform.* From left to right: The original image, the inverse Fourier transform of the Fourier transform of the original image, the inverse Fourier transform but after randomizing the amplitude of the spectrum, Guided Backprop $\phi^\dagger(\mathbf{p}, \mathbf{r})$ with positive random input ($\mathbf{p}$) and DeConvNet with positive random input ($\mathbf{p}$). Best viewed on screen.

In the Fourier transform, changing the phase of a spectral component $Ae^{j(\omega x+\theta)}$ by $\Delta\theta$ amounts to shifting it by $-\Delta\theta/\omega$. Furthermore, negating the signal is equivalent to a phase shift of π. In the deconvolutional architectures, max pooling switches record the location of filter activations, whereas the ReLUs applied in the backward direction contribute to reconstructing the polarity. More precisely, in the forward pass the ReLU block computes $y = \max\{0, x\}$. In the backward direction, the signal $\hat{y}$ is propagated towards the input as follows:

$$\hat{x} = \max\{\hat{y}, 0\} \text{ (in DeConvNet)}, \quad \hat{x} = \max\{\hat{y}, 0\} \odot [x > 0] \text{ (in Guided Backprop)}. \tag{7.11}$$

We see that both constructions guarantee that the polarity of the backward signal $\hat{x}$ is the same as the polarity of the forward signal y, which is non-negative. In fact, DeConvNet guarantees that $y\hat{x} \geqslant 0$, and Guided Backprop adds the guarantee that $y = 0 \Rightarrow \hat{x} = 0$. The first condition is stronger in term of preserving the polarity, and as seen in fig. 7.4, it is necessary to obtain a clear reconstruction of the image edges.

Note that while we make a comparison between CNNs and the fourier transform, our analysis here is very different from prior work which tries to explain neural networks based on a hierarchy of wavelet convolutions [90, 91].

	Method	CNN	%PP	%IoU
Weak/No Supervision	Network Saliency	AlexNet	82.8	57.1
	Guided Backprop	AlexNet	82.3	55.6
	DeConvNet	AlexNet	75.9	48.3
	Network Saliency	VGG-16	82.5	56.3
	Guided Backprop	VGG-16	83.3	56.3
	DeConvNet	VGG-16	76.5	48.2
	Baseline	-	79.0	46.3
	Baseline of [50]	-	73.4	24.0
	Method of [50]	-	**84.4**	**57.3**

Table 7.1: Mean Intersection over Union (IoU) and mean Per-Pixel (PP) accuracy for different segmentation methods on the dataset of [50].

7.4.5 Effect of Local Response Normalization (LRN)

To reverse LRN layers, we use BP-reversed LRN for Network Saliency and Guided Backprop and Identity for DeConvNet. For Network Saliency, this agrees with the algorithm of Simonyan *et al.* [121]. For DeConvNet, this agrees with the original architecture of Zeiler *et al.* [156]. Figure 7.10 shows that this choice does not matter much for Guided Backprop.

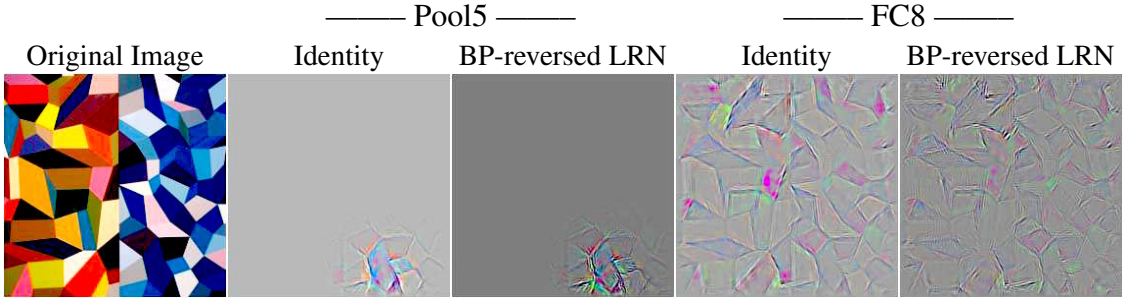


Figure 7.10: Guided Backprop visualizations of Pool5 and fc8 neurons in AlexNet while setting LRN reverse to either the identity or BP-reversed LRN. The difference is small.

7.4.6 Objectness for Free: Weakly-Supervised Salient Object Segmentation

In this section we demonstrate that pre-trained CNNs reversed using Network Saliency and Guided Backprop can segment generic foreground objects. To this end, we consider

the benchmark dataset of [50] consisting of 4276 ImageNet images annotated with the binary mask of the foreground object. Notably, the object categories in this benchmarks are partially disjoint from the ones in the ImageNet ILSVRC data used to pre-train the CNNs: of the 445 synsets present in the segmentation benchmark data only 215 of them overlap with the 1000 ILSVRC classes.

In order to perform segmentation, we improve the setup of [121]. Given an image $\mathbf{x}$, the CNN $\phi(\mathbf{x})$ is evaluated until the last layer before softmax (FC8 in AlexNet[6] and VGG-VD), recording the bottleneck information $\mathbf{r}$. Rather than resizing the image to the standard network input size, the CNN is applied in a fully convolutional manner [85]. The tensor $\mathbf{e}^*$ is set to the indicator of the channel that contains the maximally-activated neuron in FC8. Next the L^∞ norm of each RGB triplet in the output $\hat{\mathbf{x}} = \phi^\dagger(\mathbf{e}^*, \mathbf{r})$ of the reversed architecture is computed, and the resulting saliency map is used in GrabCut [51] to segment the object as in [121]. Besides the ILSVRC data used to pre-train the CNN and 98 segmented images for validating the design choices above, there is no further training. For segmentation, this is a weakly-supervised setting as no object bounding boxes or segmentations are used for training.

Table 7.1 compares the three reversed architectures and the method of [50]. The latter uses a combination of segment transfer from VOC2010 data, label propagation, bounding box annotations for 60k training images, and class label annotations for all images. We also compare against a simple baseline obtained using a fixed Gaussian blob as saliency map (fig. 7.11), similar but much better than the analogous baseline in [50]. Our baseline used a larger variance for the Gaussian distribution. See fig. 7.11 for qualitative results.

Guided Backprop and Network Saliency perform equally well, much better than the baseline, and nearly as well as the method of [50], despite using weak supervision and a training set that, for the most part, contains different classes from the test set. This suggests that CNNs learn the appearance of generic objects, which Network Saliency and Guided Backprop can extract efficiently. DeConvNet perform comparably to the Gaussian baseline confirming its lack of foreground selectivity (section 7.4.3).

[6]For Guided Backprop, the LRN layers in AlexNet are reversed using BP-reversal $\mathrm{LRN}^{\mathrm{BP}}$ instead of the identity, which was found to be slightly superior in terms of IoU performance.

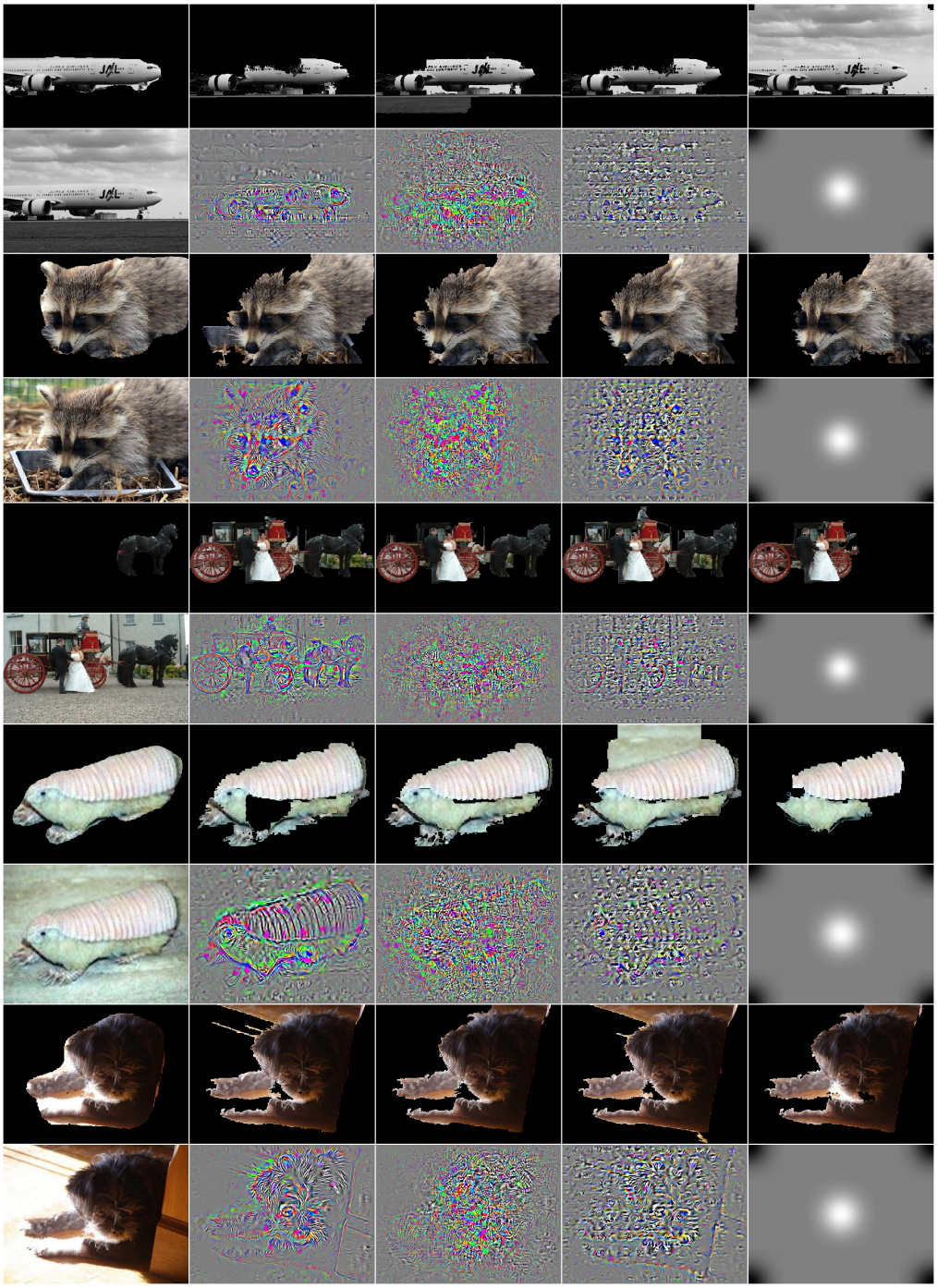


Figure 7.11: Segmentation results (random selection). For each image, the top row shows the GrabCut segmentation and the bottom row shows the output of the corresponding deconvolutional network derived from AlexNet. The latter is used to seed GrabCut.

Somewhat surprisingly, Guided Backprop did not perform better than Network Saliency, despite achieving in general much sharper saliency maps. Qualitatively, it appears that GrabCut prefers a more diffuse saliency map as opposed to a sharper one that focuses on the object boundaries, which may create "holes" in the segmentation. In fact, we hypothesize that GrabCub [51] is a key contributor to the impressive performance of the otherwise weak Gaussian baseline.

7.5 Discussion

In this chapter we derived a general construction for reversed "deconvolutional" architectures, showed that BP is an instance of such a construction, and used this to precisely contrast DeConvNet, Guided Backprop and Network Saliency. Guided Backprop produced convincingly sharper images than Network Saliency while being more selective to foreground objects than DeConvNet.

We primarily showed that bottleneck information, pooling switches and ReLU masks, dominate the output of reversed architectures which questions their utility in characterizing individual neurons. We also showed that the sharpness of generated images depends mainly on the polarization enforced by reversed ReLU units, followed by the ReLU rectification masks, and with a secondary contribution from the max pooling switches. Some of these ideas may be transferable to other applications of deconvolution such as the U-architecture of [97] for semantic segmentation.

8 Summary and Future Work

This chapter presents a brief summary of the key observations and contributions of this thesis. We also share our insights for possible future work and discuss notable recent publications that followed or concurred with our contributions to these interesting topics.

8.1 Self-Supervised Learning Using Motion Cues

We presented two methods for self-supervised learning of generic static image representations. Both of these exploited motion cues as the supervisory signal. Our methods attempt to address the key challenge of ambiguity in predicting motion from a single frame by side-stepping this prediction altogether. Instead, inspired by the Gestalt principle of common fate, they learned a CNN to either group or embed pixels based on how they might move in a video.

We presented qualitative results in the form of a visualizations of this grouping. We then presented quantitative results by transfer learning the representation into various recognition tasks. The more scalable cross pixel flow similarity model (CPFS-CE) does well in transfer learning experiments achieving the state-of-the-art among motion based self-supervised models. The S3-CNN model shows promising results on synthetic data and is perhaps usable in specific settings where certain transformation classes approximate the motion of pixels and optical-flow is accurate.

We see our contributions as an instance of self-supervision using multiple modalities, RGB and optical-flow, which poses our work as a special case of this broader area of research.

Future work: Optical-flow is effective in capturing dense but short range motion. On the other hand keypoint tracking works well for longer time spans but is sparse. We would like to experiment using the cross pixel flow similarity loss on such long term tracks. The key benefit of increased time is that a longer motion signature may reveal more about the pixel's latent object identity. For instance, long term motion might correlate with center of mass motion thus potentially encouraging the network to group pixels on different parts of a non-rigid object.

In this thesis we noted that models, pre-trained on motion based self-supervision proxy tasks, performed worse than the state-of-the-art on various recognition benchmarks - VOC 2007 classification, VOC 2007 detection and ILSVRC-2012 linear probing. This performance gap was widened by a recent clustering based unsupervised method [12] which does not rely on any extra modality or cue. This gap needs to be addressed by future work.

Another promising direction is the use of contrastive predictive coding [104] to learn from sequences in general. They exploit spatial context to form a sequence within static images instead of using videos, and get high classification accuracy by transfer learning. It is not clear whether their performance should be attributed to a better exploitation of sequence statistics or to the use of non-overlapping image context. Their method effectively maximizes the mutual information between a prediction of the future and a latent encoding of the same. Mutual information has also been used in [69] to learn semantics in an unsupervised manner.

8.2 Visualizing CNNs using Natural Pre-Images

There is a growing interest in methods that can help us understand computer vision representations, and in particular those learned automatically from data. In this thesis we have presented a suite of visualization methods using natural pre-images - inversion,

activation maximization, and caricaturization. All three methods are unified into one regularized energy minimization framework. The regularization biases the pre-images to look natural and thus interpretable. The core loss function makes the pre-images useful in different ways: *Inversion*, the pre-image is such that its representation matches a target as closely as possible; *neuron maximization*, the pre-image maximally activates individual feature components; *caricaturization*, the pre-image enhances the positive activations across multiple representation components. Thus all three methods are complementary.

We used them to probe and compare standard classical representations, and CNNs. An analysis of the visualizations revealed properties of CNNs: photometrically accurate information is preserved deep down in CNNs; even very deep layers contain instance-specific information about objects; intermediate convolutional layers capture local invariances to pose and fully connected layers to large variations in object layout; individual CNN components code for complex but, for the most part, not semantically-obvious patterns; different CNN layers appear to capture different types of structures in images, from lines and curves to parts.

The robustness of our visualization by inversion method was assessed quantitatively and the usefulness of our regularization confirmed using a user study.

Future work: We refer readers to [103] for a summary of publications following our early contribution to this growing community of feature visualization. More recently deep neural networks have themselves been used as natural image priors [133, 95]. These deep priors are very expressive and may influence the visualizations more than merely making them natural, thus clouding our understanding of the pre-trained network. Nonetheless, this line of visualization methods may be used as direct diagnostic tools to further research in CNNs. For example, when learning from multiple modalities, inversion may be used to check whether or not all modalities are captured by a joint representation. Similarly, in such applications, activation maximization can be used to identify neurons that span multiple input modalities. That said, many open questions remain. Visualization methods presented in this thesis do not provide a functional understanding of CNNs. In other words, while we can visualize *what* is captured or preferred by a representation, we lack

an abstract understanding of *how* such a representation is computed. Future work on visualizing CNNs should attempt to address this and related aspects of CNN learning.

8.3 Visualizations of CPFS-CE

We applied our visualization suite to the self-supervised CPFS-CE model of chapter 3 and compared it against the visualizations of an ILSVRC-12 pre-trained model and a randomly initialized one. We observed that our model and the fully supervised one have more structure in their visualizations. We also noticed a face-like neuron in the fifth convolutional layer of our self-supervised model, which we hypothesized is a result of many people centric videos in the YFCC100m dataset. The ILSVRC-12 pre-trained model, however, had more localized structure resembling various common objects suggesting that it captures more semantics. This semantic gap was verified using the Network Dissection approach. We observed that most neurons in the conv5 layer of our CPFS-CE pre-trained model were selective of low-level textures. The bias for face and face-parts was evident here as well.

8.4 Salient Deconvolutional Networks

We presented a critique of gradient based neuron visualization methods namely DeConvNets, Guided Backprop and Network Saliency. We showed that their output is not neuron specific and is largely influenced by auxiliary information particularly for deep layers. Thus their utility in visualizing individual neurons is unclear. We identified aspects of their heuristics that contribute to the structure and localization in their output namely the setting of pooling switches and the ReLU rectification mask. We discussed how these might relate to "phase" information.

Following DeConvNets, Network Saliency and Guided Backprop, a number of new methods have been proposed to highlight salient regions corresponding to individual neurons [160, 4, 118, 164]. We note that Excitation Backprop [160] largely address the lack of neuron specificity in gradient based saliency methods using a contrastive probabilistic

winner take all mechanism. It is, however, unclear as to how their approach overcomes the influence of auxiliary information which is still an open question for future work.

Appendices

A

AlexNet Specification for CPFS-CE Model

```
name: "ALEXNET"
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 224
  dim: 224
}

# ==== Network Definition Starts ==========

# Custom python layer below with param_str:
#     [top, bottom, left, right]
# This padding is meant to replicate the tensorflow
# convolution/pooling in caffe.
layer {
  name: "pad_data"
  type: "Python"
  bottom: "data"
  top: "padded_data"
  python_param {
    param_str: "[3, 4, 3, 4]"
    module: "pad_layer"
    layer: "PadLayer"
  }
}
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "padded_data"
  top: "conv1"
  convolution_param {
    engine: CAFFE
    num_output: 96
    pad_h: 0
    pad_w: 0
    kernel_h: 11
    kernel_w: 11
    stride_h: 4
    stride_w: 4
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pad_conv1"
  type: "Python"
  bottom: "conv1"
  top: "conv1_padded"
  python_param {
    param_str: "[0, 1, 0, 1]"
    module: "pad_layer"
    layer: "PadLayer"
  }
}
layer {
  name: "maxpool1"
  type: "Pooling"
  bottom: "conv1_padded"
  top: "maxpool1"
  pooling_param {
    pool: MAX
    kernel_h: 3
    kernel_w: 3
    stride_h: 2
    stride_w: 2
    pad_h: 0
```

```
    pad_w: 0
  }
}
layer {
  name: "lrn1"
  type: "LRN"
  bottom: "maxpool1"
  top: "lrn1"
  lrn_param {
    local_size: 5
    alpha: 0.0005
    beta: 0.75
    k: 1.0
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "lrn1"
  top: "conv2"
  convolution_param {
    engine: CAFFE
    num_output: 256
    pad_h: 2
    pad_w: 2
    kernel_h: 5
    kernel_w: 5
    stride_h: 1
    stride_w: 1
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pad_conv2"
  type: "Python"
  bottom: "conv2"
  top: "conv2_padded"
  python_param {
    param_str: "[0, 1, 0, 1]"
    module: "pad_layer"
```

```
    layer: "PadLayer"
  }
}
layer {
  name: "maxpool2"
  type: "Pooling"
  bottom: "conv2_padded"
  top: "maxpool2"
  pooling_param {
    pool: MAX
    kernel_h: 3
    kernel_w: 3
    stride_h: 2
    stride_w: 2
    pad_h: 0
    pad_w: 0
  }
}
layer {
  name: "lrn2"
  type: "LRN"
  bottom: "maxpool2"
  top: "lrn2"
  lrn_param {
    local_size: 5
    alpha: 0.0005
    beta: 0.75
    k: 1.0
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "lrn2"
  top: "conv3"
  convolution_param {
    engine: CAFFE
    num_output: 384
    pad_h: 1
    pad_w: 1
    kernel_h: 3
    kernel_w: 3
    stride_h: 1
    stride_w: 1
  }
```

```
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    engine: CAFFE
    num_output: 384
    pad_h: 1
    pad_w: 1
    kernel_h: 3
    kernel_w: 3
    stride_h: 1
    stride_w: 1
  }
}
layer {
  name: "relu4"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
}
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  convolution_param {
    engine: CAFFE
    num_output: 256
    pad_h: 1
    pad_w: 1
    kernel_h: 3
    kernel_w: 3
    stride_h: 1
    stride_w: 1
  }
}
```

```
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "conv5"
  top: "conv5"
}

# ==== Network Definition Until Conv5 ==========

layer {
  name: "pad_conv5"
  type: "Python"
  bottom: "conv5"
  top: "conv5_padded"
  python_param {
    param_str: "[0, 1, 0, 1]"
    module: "pad_layer"
    layer: "PadLayer"
  }
}
layer {
  name: "maxpool5"
  type: "Pooling"
  bottom: "conv5_padded"
  top: "maxpool5"
  pooling_param {
    pool: MAX
    kernel_h: 3
    kernel_w: 3
    stride_h: 2
    stride_w: 2
    pad_h: 0
    pad_w: 0
  }
}

# ==== Network Definition FC Layers ==========

layer {
  name: "fc6"
  type: "Convolution"
  bottom: "maxpool5"
  top: "fc6"
  convolution_param {
    engine: CAFFE
```

```
    num_output: 256
    pad_h: 3
    pad_w: 3
    kernel_h: 7
    kernel_w: 7
    stride_h: 1
    stride_w: 1
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "fc6"
  top: "fc6"
}
layer {
  name: "caffe.Dropout_18"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  exclude {
    phase: TEST
  }
  dropout_param {
    dropout_ratio: 0.5
  }
}
layer {
  name: "fc7"
  type: "Convolution"
  bottom: "fc6"
  top: "fc7"
  convolution_param {
    engine: CAFFE
    num_output: 256
    pad_h: 0
    pad_w: 0
    kernel_h: 1
    kernel_w: 1
    stride_h: 1
    stride_w: 1
  }
}
layer {
  name: "relu7"
```

```
  type: "ReLU"
  bottom: "fc7"
  top: "fc7"
}
layer {
  name: "caffe.Dropout_21"
  type: "Dropout"
  bottom: "fc7"
  top: "fc7"
  exclude {
    phase: TEST
  }
  dropout_param {
    dropout_ratio: 0.5
  }
}

# ==== Network Definition Ends ==========

# The sparse-hypercolumn is implemented in tensorflow.
# It combines blobs - "conv1", "pool1", "conv3",
#                     "pool5", "fc7"

# The MLP mapping it into the 16-D embedding is below

layer {
  name: "embed_fc1"
  type: "Convolution"
  bottom: "hypercolumn"
  top: "embed_fc1"
  convolution_param {
    engine: CAFFE
    num_output: 1024
    pad_h: 0
    pad_w: 0
    kernel_h: 1
    kernel_w: 1
    stride_h: 1
    stride_w: 1
  }
}
layer {
  name: "embed_fc1_relu"
  type: "ReLU"
  bottom: "embed_fc1"
```

```
  top: "embed_fc1"
}


layer {
  name: "embed_fc2"
  type: "Convolution"
  bottom: "embed_fc1"
  top: "embed_fc2"
  convolution_param {
    engine: CAFFE
    num_output: 16
    pad_h: 0
    pad_w: 0
    kernel_h: 1
    kernel_w: 1
    stride_h: 1
    stride_w: 1
  }
}

# This is followed by a L2 normalize layer in tensorflow.
```

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the International Conference on Computer Vision*, pages 37–45. IEEE, 2015.

[3] Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. In *Proceedings of the International Conference on Computer Vision*, pages 609–617, 2017.

[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*, 10(7), 2015.

[5] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *Proceedings of the International Conference on Computer Vision*, 2015.

[6] Aayush Bansal, Xinlei Chen, Bryan Russell, Abhinav Gupta, and Deva Ramanan. PixelNet: Representation of the pixels, by the pixels, and for the pixels. *arXiv:1702.06506*, 2017.

[7] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network Dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[8] Yoshua Bengio and Olivier Delalleau. On the expressive power of deep architectures. In *Algorithmic Learning Theory*, pages 18–36. Springer Berlin Heidelberg, 2011.

[9] Christopher M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[10] Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. In *Proceedings of the International Conference on Machine Learning*, pages 517–526. PMLR, 2017.

[11] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Proceedings of the European Conference on Computer Vision*, pages 611–625. Springer, 2014.

[12] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *arXiv preprint arXiv:1807.05520*, 2018.

[13] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proceedings of the British Machine Vision Conference*, 2014.

[14] Yunjin Chen, René Ranftl, and Thomas Pock. A bi-level view of inpainting-based image compression. In *Proc. of Computer Vision Winter Workshop*, 2014.

[15] Nello Cristianini and J Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[16] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[17] Navneet Dalal and Bill Triggs. Histogram of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 886–893, 2005.

[18] Emmanuel d'Angelo, Alexandre Alahi, and Pierre Vandergheynst. Beyond bits: Reconstructing images from local binary descriptors. In *Proceedings of the International Conference on Pattern Recognition*, pages 935–938, Nov 2012.

[19] Emmanuel d'Angelo, Laurent Jacques, Alexandre Alahi, and Pierre Vandergheynst. From bits to images: Inversion of local binary descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):874–887, May 2014.

[20] Virginia R de Sa. Learning classification with unlabeled data. In *Advances in Neural Information Processing Systems*, pages 112–119, 1994.

[21] Andrew Delong, Anton Osokin, Hossam Isack, and Yuri Boykov. Fast approximate energy minimization with label costs. *International Journal of Computer Vision*, 96:1–27, Jan. 2012.

[22] Emily L Denton and Vighnesh Birodkar. Unsupervised learning of disentangled representations from video. In *Advances in Neural Information Processing Systems*, pages 4414–4423, 2017.

[23] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the International Conference on Computer Vision*, pages 1422–1430, 2015.

[24] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the International Conference on Computer Vision*, 2017.

[25] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *International Conference on Learning Representations*, 2017.

[26] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4829–4837, 2016.

[27] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the International Conference on Computer Vision*, pages 2758–2766, 2015.

[28] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1734–1747, 2016.

[29] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[30] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009.

[31] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[32] Mark Everingham, Luc Van Gool, Christoper K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007.

[33] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[34] Alon Faktor and Michal Irani. Video segmentation by non-local consensus voting. In *Proceedings of the British Machine Vision Conference*, volume 2, page 8, 2014.

[35] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[36] Basura Fernando, Hakan Bilen, Efstratios Gavves, and Stephen Gould. Self-supervised video representation learning with odd-one-out networks. In *Proceedings of the International Conference on Computer Vision*, 2017.

[37] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[38] 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*, 2018.

[39] Chuang Gan, Boqing Gong, Kun Liu, Hao Su, and Leonidas J Guibas. Geometry guided convolutional neural networks for self-supervised video representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[40] Rouhan Gao, Dinesh Jayaraman, and Kristen Grauman. Object-centric representation learning from unlabeled videos. In *Proceedings of the Asian Conference on Computer Vision*, 2016.

[41] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[42] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[43] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI Vision Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[44] Spyros Gidaris, Praveen Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[45] Ross B. Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision*, pages 1440–1448, 2015.

[46] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[47] Ross B Girshick, Pedro F Felzenszwalb, and David McAllester. Discriminatively trained deformable part models, release 5. `http://people.cs.uchicago.edu/~rbg/latent-release5/`.

[48] Mike Goslin and Mark R Mine. The Panda3D graphics engine. *Computer*, 37(10):112–114, Oct. 2004.

[49] Hayit Greenspan, Bram Van Ginneken, and Ronald M Summers. Guest editorial Deep Learning in Medical Imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5):1153–1159, 2016.

[50] Matthieu Guillaumin, Daniel Küttel, and Vittorio Ferrari. ImageNet auto-annotation with segmentation propagation. *International Journal of Computer Vision*, 110(3):328–348, 2014.

[51] Varun Gulshan, Carsten Rother, Antonio Criminisi, Andrew Blake, and Andrew Zisserman. Geodesic star convexity for interactive image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3129–3136, 2010.

[52] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Proceedings of the International Conference on Computer Vision*, pages 991–998, 11 2011.

[53] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[54] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[55] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the International Conference on Computer Vision*, pages 2980–2988, 2017.

[56] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 2006.

[57] Seunghoon Hong, Hyeonwoo Noh, and Bohyung Han. Decoupled deep neural network for semi-supervised semantic segmentation. In *Advances in Neural Information Processing Systems*, 2015.

[58] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[59] Jinggang Huang and David Mumford. Statistics of natural images and models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1541–1547, 1999.

[60] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, 2015.

[61] Hossam Isack and Yuri Boykov. Energy-based geometric multi-model fitting. *International Journal of Computer Vision*, 97:123–147, April 2012.

[62] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Learning visual groups from co-occurrences in space and time. In *International Conference on Learning Representations*, 2015.

[63] Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *Proceedings of the International Conference on Computer Vision*, 2015.

[64] Dinesh Jayaraman and Kristen Grauman. Slow and steady feature analysis: higher order temporal coherence in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[65] Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to egomotion from unlabeled video. *International Journal of Computer Vision*, 125:136–161, 2017.

[66] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3304–3311. IEEE, 2010.

[67] Simon Jenni and Paolo Favaro. Self-supervised feature learning by learning to spot artifacts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[68] Craig A Jensen, Russell D Reed, Robert J Marks, Mohamed A El-Sharkawi, Jae-Byung Jung, Robert T Miyamoto, Gregory M Anderson, and Christian J Eggen. Inversion of feedforward neural networks: algorithms and applications. *Proc. of the IEEE*, 87(9), Sept 1999.

[69] Xu Ji, João Henriques, and Andrea Vedaldi. Invariant information distillation for unsupervised image segmentation and clustering. *arXiv:1807.06653*, 2018.

[70] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[71] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290:91–97, march 1981.

[72] Hiroharu Kato and Tatsuya Harada. Image reconstruction from bag-of-visual-words. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2014.

[73] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[74] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *International Conference on Learning Representations*, 2016.

[75] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, pages 1033–1041, 2009.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[77] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In *Proceedings of the European Conference on Computer Vision*, pages 577–593. Springer, 2016.

[78] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Colorization as a proxy task for visual understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[79] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *Proceedings of the International Conference on Machine Learning*, 2012.

[80] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Kumar Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequence. In *Proceedings of the International Conference on Computer Vision*, 2017.

[81] S. Lee and R. M. Kil. Inverse mapping of continuous functions using local and global information. *IEEE Trans. on Neural Networks*, 5(3), 1994.

[82] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, June 2001.

[83] Alexander Linden and J. Kindermann. Inversion of multilayer nets. In *Proc. International Conference on Neural Networks*, 1989.

[84] Ce Liu. *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis*. PhD thesis, Massachusetts Institute of Technology, USA, 2009.

[85] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[86] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[87] David Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th International Conference on Computer Vision, Kerkyra, Greece*, pages 1150–1157, September 1999.

[88] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[89] Bao-Liang Lu, Hajime Kita, and Yoshikazu Nishikawa. Inverting feedforward neural networks using linear and nonlinear programming. *IEEE Trans. on Neural Networks*, 10(6), 1999.

[90] Stéphane Mallat. Recursive interferometric representation. In *Proc. of EUSICO conference, Denmark*, 2010.

[91] Stéphane Mallat. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

[92] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Shuffle and learn: Unsupervised learning using temporal order verification. In *Proceedings of the European Conference on Computer Vision*, pages 527–544, 2016.

[93] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. `http://googleresearch.blogspot.co.uk/2015/06/inceptionism-going-deeper-into-neural.html`, 2015.

[94] T Nathan Mundhenk, Daniel Ho, and Barry Y. Chen. Improvements to context based self-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[95] 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 *Advances in Neural Information Processing Systems*, 2016.

[96] Anh Nguyen, Jason Yosinski, and Jeff 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.

[97] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the International Conference on Computer Vision*, 2015.

[98] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *Proceedings of the European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[99] Mehdi Noroozi, Hamed Pirsiavash, and Paolo Favaro. Representation learning by learning to count. In *Proceedings of the International Conference on Computer Vision*, pages 5898–5906, 2017.

[100] Mehdi Noroozi, Ananth Vinjimoor, Paolo Favaro, and Hamed Pirsiavash. Boosting self-supervised learning via knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[101] David Novotny, Sam Albanie, Diane Larlus, and Andrea Vedaldi. Self-supervised learning of geometrically stable features through probabilistic introspection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[102] Eric Nowak, Frédéric Jurie, and Bill Triggs. Sampling strategies for bag-of-features image classification. In *Proceedings of the European Conference on Computer Vision*, pages 490–503, 2006.

[103] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[104] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[105] Andrew Owens, Phillip Isola, Josh H. McDermott, Antonio Torralba, Edward H. Adelson, and William T. Freeman. Visually indicated sounds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2405–2413, 2016.

[106] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[107] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[108] Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[109] Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*. 2012.

[110] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000.

[111] Alessandro Prest, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Learning object class detectors from weakly annotated video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3282–3289, 2012.

[112] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015.

[113] Zhongzheng Ren and Yong Jae Lee. Cross-domain self-supervised multi-task feature learning using synthetic imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[114] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[115] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Deepmatching: Hierarchical deformable dense matching. *International Journal of Computer Vision*, 120:300–323, 2016.

[116] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[117] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[118] 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 *Proceedings of the International Conference on Computer Vision*, pages 618–626, 2017.

[119] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations*, 2014.

[120] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. In *Proceedings of the International Conference on Robotics and Automation*, 2018.

[121] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

[122] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1573–1585, 2014.

[123] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[124] Josef Sivic, Frederik Schaffalitzky, and Andrew Zisserman. Object level grouping for video shots. *International Journal of Computer Vision*, 67(2):189–210, 2006.

[125] Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the 9th International Conference on Computer Vision, Nice, France*, volume 2, pages 1470–1477, 2003.

[126] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations Workshop*, 2015.

[127] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using LSTMs. In *Proceedings of the International Conference on Machine Learning*, volume 37, 2015.

[128] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[129] Aditya Tatu, Francois Lauze, Mads Nielsen, and Benjamin Kimia. Exploring the representation capabilities of the HOG descriptor. In *International Conference on Computer Vision Workshop*, 2011.

[130] James Thewlis. Panda3D optical flow. `https://github.com/jamt9000/pandaflowgen`.

[131] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: the new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[132] Dejan Todorovic. Gestalt principles. *Scholarpedia*, 3(12):5345, 2008. revision #91314.

[133] Dmitry Ulyanov and Andrea Vedaldi. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[134] Annamária R. Várkonyi-Kóczy and A. Rövid. Observer based iterative neural network model inversion. In *IEEE International Conference on Fuzzy Systems*, 2005.

[135] Andrea Vedaldi. An open implementation of the SIFT detector and descriptor. Technical Report 070012, UCLA CSD, 2007.

[136] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the ACM Multimedia Conference*, 2015.

[137] Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, and Antonio Torralba. HOGgles: Visualizing object detection features. In *Proceedings of the International Conference on Computer Vision*, pages 1–8, 2013.

[138] Carl Vondrick, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy. Tracking emerges by colorizing videos. In *Proceedings of the European Conference on Computer Vision*, 2018.

[139] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *Proceedings of the European Conference on Computer Vision*, pages 835–851, 2016.

[140] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[141] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *Proceedings of the International Conference on Computer Vision*, 2015.

[142] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In *Proceedings of the International Conference on Computer Vision*, 2017.

[143] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3360–3367, 2010.

[144] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the International Conference on Computer Vision*, pages 2794–2802, 2015.

[145] Xiaolong Wang, Kaiming He, and Abhinav Gupta. Transitive invariance for self-supervised visual representation learning. In *Proceedings of the International Conference on Computer Vision*, 2017.

[146] Andreas Wedel, Annemarie Meißner, Clemens Rabe, Uwe Franke, and Daniel Cremers. Detection and segmentation of independently moving objects from dense scene flow. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 14–27, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[147] Donglai Wei, Joseph Lim, Andrew Zisserman, and William T. Freeman. Learning and using the arrow of time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[148] Philippe Weinzaepfel, Herv'e Jégou, and Patrick Pérez. Reconstructing an image from its local descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 337–344, 2011.

[149] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. DeepFlow: Large displacement optical flow with deep matching. In *Proceedings of the International Conference on Computer Vision*, pages 1385–1392, 2013.

[150] Ronald J. Williams. Inverting a connectionist network mapping by back-propagation of error. In *Proc. CogSci*, 1986.

[151] Tianfan Xue, Jiajun Wu, Katherine L Bouman, and William T Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2016.

[152] Tianfan Xue, Jiajun Wu, Katherine L Bouman, and William T Freeman. Visual dynamics: Stochastic future generation via layered cross convolutional networks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[153] Jianchao Yang, Kai Yu, and Thomas Huang. Supervised translation-invariant sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3517–3524, 2010.

[154] Jason Yosinksi, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *International Conference on Machine Learning Workshop*, 2015.

[155] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method. *CoRR, arXiv:1212.5701*, 2012.

[156] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833, 2014.

[157] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[158] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Proceedings of the International Conference on Computer Vision*, pages 2018–2025, 2011.

[159] Xiaohang Zhan, Ziwei Liu, Ping Luo, Xiaoou Tang, and Chen Change Loy. Mix-and-match tuning for self-supervised semantic segmentation. In *AAAI Conference on Artificial Intelligence*, February 2018.

[160] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *Proceedings of the European Conference on Computer Vision*, pages 543–559. Springer, 2016.

[161] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *Proceedings of the European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[162] Richard Zhang, Phillip Isola, and Alexei A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[163] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *arXiv:1711.05611*, 2018.

[164] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[165] Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *Proceedings of the European Conference on Computer Vision*, pages 141–154, 2010.

[166] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[167] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random-fields and maximum-entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, March 1998.