Compression of Models and Data in Deep Learning

Milad Alizadeh

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

تقدیم به مادر و پدرم،
ویدا و امین

To my parents,
Vida and Amin

Acknowledgements

I would like to thank my supervisors, Nic and Yarin, for giving me the opportunity to join their new labs in Oxford and Cambridge, and for their invaluable inspiration, insight, and motivation over the years. This opportunity changed my life in more ways than they can imagine.

I am also grateful to my friends, collaborators, and mentors during my studies, particularly: Arash Behboodi, Tijmen Blankevoort, Tim de Bruin, Emilien Dupont, Sebastian Farquhar, Javier Fernandez-Marques, Adam Goliński, Aidan Gomez, Andrew Jesson, Christos Louizos, Hrushikesh Loya, Markus Nagel, Tim Rudner, Lisa Schut, Lewis Smith, Shyam Tailor, Panagiotis Tigas, Jakub Tomczak, Catherine Tong, Joost van Amersfoort, Mart van Baalen, and Luisa Zintgraf.

I want to thank my parents, Vida and Amin, and my sisters Yalda and Ladan – without them the work in this thesis would not have been possible. Through ups and downs, I always knew I had a home to go back to.

Finally, I would like to thank Fuchsia for her kindness, love, and making this journey worthwhile.

Abstract

We face many challenges in deploying high-performance neural networks in practice. These challenges are predominantly due to the size of neural networks and apply to both training and inference. Compressing neural networks to make them train and run more *efficiently* is therefore crucial and has been a parallel line of research from the early days of neural networks development. The two main compression techniques in deep learning, which are the focus of this thesis, are pruning and quantization. This thesis explores how the information from higher-order gradients (meta-gradients) be used to improve deep learning compression. We start by identifying a fundamental limitation in the formulation of pruning: Although many methods, such as saliency-based pruning, follow pruning by a training or fine-tuning stage, parameter saliencies only look at a snapshot of parameters without taking into account the *trainability* of the parameters. We show how meta-gradients can be used as a more informative signal to find better trainable subnetworks at initialization. We then look at quantized neural networks and show how meta-gradients can be used in a regularization scheme to *learn* models with inherent robustness against post-training quantization. Finally, we look at the dual compression problem, i.e. using neural networks to compress *data* sources. We start with images and propose a simple *autoencoder-free* architecture where we store weights of a neural network instead of RGB values of image pixels. We then use meta-gradients to *meta-learn* a base network to amortize the cost of training one network per input. A significant advantage of our *learning* compression is that it becomes agnostic to the data type, and we present results on various data types beyond 2D images. Importantly, we evaluate the usefulness of standard DNN compression techniques, e.g., quantization, for this new type of neural network.

Contents

Chapter 1

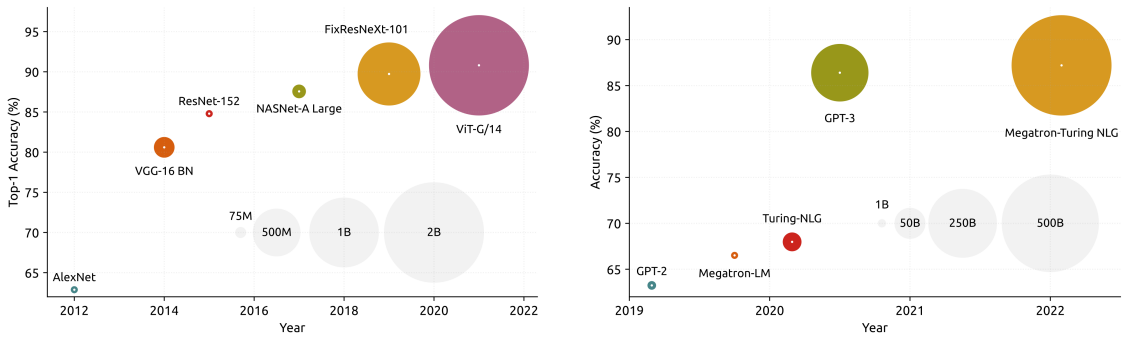
Introduction

1.1 Deep Learning and Compression

The success of deep learning in recent years was made possible by several breakthroughs and paradigm shifts that happened around the same time: adoption of back-propagation for training, advancements in convolutional neural networks, availability of large-scale datasets, and crucially, a massive increase in computation power due to GPUs evolving into general-purpose, highly-parallel, multithreaded, computing platforms. With the continuous progress of algorithms and computing power, training large neural networks that can utilize enormous amounts of data and make remarkably valuable predictions across a wide range of applications is now well within our reach.

However, as we enter an era where we wish to use high-performance neural networks in more and more aspects of our lives, we face many challenges in deploying them in practice. These challenges are predominantly due to the size of neural networks. It is now easy for a neural network to have millions or billions of parameters, and there is no sign that we are slowing down, particularly in some applications such as natural language processing where we have not seen the end of the scaling law. This can be seen in Figure 1.1, which shows the trend of model size and performance in two particular vision and natural language processing tasks.

The challenges imposed by the size of neural networks apply to both training



(a) Image Classification on ImageNet ReaL

(b) Language Modelling on LAMBADA

Figure 1.1: Examples of the significant increase in neural networks' size over time in two tasks in vision and language modeling. The area of the circles is proportional to the number of network parameters. A legend is provided in the bottom right corners. The figure is inspired by Canziani et al. (2016)

and inference. On the inference side, there is an increased desire to run models privately on-device. Some example applications are speech-to-text recognition and auto-captioning, where the models must run on-device and in real-time. The devices that need to run such applications are often mobile phones, embedded systems, and IoT devices which are exactly the type of platforms that have the most stringent requirements in terms of memory, compute, latency, and energy consumption.

In addition to inference, training is also hindered by the large size of neural networks and the datasets. Firstly, training requires significantly more memory compared to inference. This is because during training we need to hold on to intermediate values generated by the network in order to complete backpropagation. This also makes parallelization difficult because we cannot do a fresh forward pass until gradients find their way all the way back to the first layer. Secondly, training large neural networks is more brittle and unstable than smaller models. Zhang et al. (2022) recently released a 175-billion parameter language model, and in addition to the model parameters, they also included the training logbook which shows that training such large models involves a great deal of engineering. As we will see in later chapters, this has far-reaching implications for efficiency approaches that require retraining or targeting specific platforms.

Compressing neural networks to make them train and run more *efficiently* is therefore crucial and has been a parallel line of research from the early days of neural networks development (LeCun et al., 1990a; Hassibi et al., 1993b). Before introducing various

approaches used to make neural networks efficient, it is essential to quickly review the main metrics we care about during the process (Reagen et al., 2017; Sze et al., 2020):

- **Accuracy or model performance.** This metric determines how successful we are in making a model efficient without compromising its performance. Accuracy is an appropriate metric for class-balanced classification tasks, which most experiments in this thesis are, and in other tasks, it can be replaced with other proxies for the performance of the model.
- **Energy and power.** Energy consumption represents the amount of data that can be processed per unit of energy and becomes crucial when running on battery-powered devices. It is a primary factor in determining the form factor of the device. Power consumption, on the other hand, is the amount of energy used per unit of time and dictates the cooling requirements of the device. It is relevant to both mobile devices and server installations.
- **Storage and memory.** Requiring more storage and memory is the first challenge caused by the size of neural networks. It is a major factor in determining the silicon chip area required for the model.
- **Throughput and latency.** In addition to storing weights and intermediate values we need to move data around fast enough. Latency measures this requirement and is defined as the time between the input data arriving to the model and when the result is available at the output. It determines whether the model can run in real-time which is crucial in many applications such as robotics, autonomous driving, and games.

Making neural networks efficient for these metrics requires optimizations at every stack level. On one end of the stack is the hardware platform used for training and running models. The continuous progress in the semiconductor fabrication process has resulted in more transistor-dense and hence more powerful and energy-efficient chips, but deep learning workloads have specifically influenced the hardware design of CPUs and GPUs and have also given rise to custom accelerators specifically targeting neural networks.

The most fundamental and taxing operation in main neural network blocks, i.e. convolutional, fully connected, and attention layers, is Multiply-And-Accumulate

(MAC). Hardware-level optimizations mostly hinge on identifying opportunities for compute *parallelism* and data *reuse* for optimizing MACs. CPUs and GPUs aim to parallelize MACs using Single Instruction, Multiple Data (SIMD) or Single Instruction, Multiple Threads (SIMT) approaches. In contrast, accelerators often take advantage of data reuse opportunities by optimizing the *order* of operation and the resulting data movement. Additionally, careful partitioning of multiplications, known as *Tiling*, becomes an important design problem when MACs are mapped to the hardware.

On the other end of the stack is the data going into the model. One example of optimizations at this level is *conditional computing* which works selectively activating *sub-parts* of the neural network at a time depending on the input of the model (Bengio et al., 2013a).

But the biggest gains can be obtained by optimizations at the top of the stack i.e. the neural networks themselves. These methods range from designing more efficient models from scratch to tuning existing models. A common technique in this field is Neural Architecture Search (NAS) where the architecture of the neural network is formulated as an optimization problem and we directly optimize for more efficient architectures. But while one could argue that existing architectures are massively over-parametrized and inefficient, nonetheless the entire ecosystem of architectures, benchmarks, optimizers, network layers, and training techniques have all been developed around having over-parameterized neural networks. Hence, the most important and common techniques for making networks efficient focus on *compressing* existing architecture into more efficient ones. The two main compression techniques in deep learning, which are the focus of this thesis, are getting rid of unimportant weights (known as *pruning*) and using lower-precision representations for model parameters and intermediate values (known as *quantization*).

Finally, another axis where compression shows up in deep learning is neural data compression which is the application of neural networks and other learning methods to *data* compression. Many parallels can be drawn between current best-of-class compression techniques and what vision and language used to do before deep learning. Areas that deep learning initially saw success in (such as image classification) were heavily based on hand-tuned engineered features tailored for specific tasks. The same thing is true for many current data compression methods, JPEG for instance,

works by making all various assumptions about the distribution of images.

Using neural networks for data compression is an old idea (Sonehara et al., 1989; Sicuranza et al., 1990) but it has improved significanlty in recent years thanks to advances in generative modeling. Many of the more recent neural compression methods heavily rely on encoder-decoder architectures (Ballé et al., 2018b; Minnen et al., 2018b; Lee et al., 2019b) that are specialized to particular applications and data distributions. Data-agnostic deep learning methods for compression is an area where neural networks can have a real impact in the next decade.

1.2 Contributions and Thesis Outline

The main contribution of this thesis is studying the applications of meta-gradients for various compression tasks. We mainly focus on neural networks in vision applications and address the following central question:

Research Question: Model Compression

Can the information from higher-order gradients be used to improve compressing deep neural networks with quantization and pruning?

In addition to studying compression *of* neural networks we also look at the dual compression problem, i.e. compressing data *with* neural networks, and address the following questions:

Research Question: Data Compression

Can the parameters of neural networks be used for storing and compressing data? If yes, are quantization and pruning techniques still applicable in this setup?

Apart from Chapter 2 which provides the necessary background material and Chapter 6 which concludes the work, the main material of this thesis is broken into three chapters (as summarized in Figure 1.2), each focusing on one compression task:

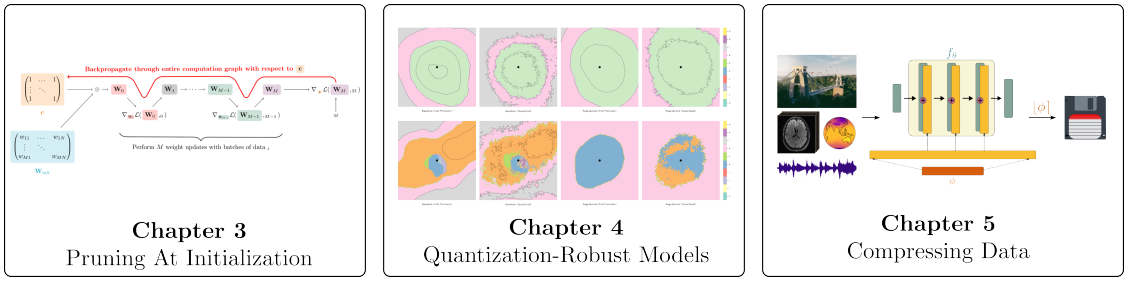


Figure 1.2: Summary of the contributions in this thesis. We start by looking at using meta-gradients as a *signal* to do pruning at initialization. We then look at how meta-gradients can be used to *regularize* training towards more quantization-robust models. Finally, we look at the problem of compressing data using neural networks and the role meta-gradients and quantization play in that setup.

Chapter 3. We begin by looking at the problem of pruning deep neural networks and propose using meta-gradients as a *signal* to make better-informed pruning decisions. Pruning has been historically performed as a post-training step and is often combined with one or multiple fine-tuning steps. The ultimate goal of this regime is to find smaller models for the inference/deployment time. There has been recent interest in the research community in performing the pruning as soon as parameters are initialized i.e. before training even begins. Pruning at initialization is desirable because the benefits of pruning in terms of computation speed-ups and memory savings can also be reaped for training. We have seen a continuous increase in the performance of this type of pruning, but there is still a performance gap compared to post-training pruning methods. In this chapter, we identify a fundamental limitation in the formulation of these methods: their saliency criteria only look at a snapshot of parameters and the loss function without taking into account the *trainability* of the model. We use meta-gradients over the few steps of the training to determine which parameters to prune and posit that this objective is more aligned with the ideal pruning objective we are after.

Chapter 4. We then shift our focus to quantization which, along with pruning, make up the two most popular approaches for compressing deep neural networks. We begin by studying some of the existing ad hoc methods that have successfully trained extremely quantized neural networks from scratch and identify common trends and techniques that push their training towards a long random search rather than following a meaningful signal. In this chapter we argue that instead of targeting

specific bit-widths and training from scratch, it would be desirable to train models that are inherently more robust to quantization and exhibit graceful performance degradation as quantization gets more extreme. We then do a principled first-order robustness analysis and show how meta-gradients can be used to modify the training objective to drive the training towards learning models that are inherently more robust against post-training quantization. To wrap up we discuss and explore possible ways of going beyond first-order robustness and the challenges they introduce.

Chapter 5. In this chapter we look at the dual compression problem, i.e. using neural networks to compress various data sources with different modalities. We start with images and propose storing the network weights instead of storing RGB values for pixels of the image. Such a network would take pixel positions as inputs and output the image's RGB values. We show how to train such a network and then use meta-gradients again to *meta-learn* a base model to amortize the cost of training a network per input. A significant benefit of *learning* compression is that it becomes agnostic to the data type, and we present results on various data types beyond 2D images. Importantly, we evaluate the usefulness of common DNN compression techniques, e.g., quantization, for this new type of neural network.

1.3 Publications and Acknowledgements

Below I provide a list of publications whose content contributed to this thesis and specifically clarify my contributions to each. Where relevant the * denotes equal contributions:

1. Joost van Amersfoort*, **Milad Alizadeh***, Sebastian Farquhar*, Nicholas Lane, and Yarin Gal. "Single Shot Structured Pruning Before Training." preprint 2020.

2. **Milad Alizadeh**, Shyam A. Tailor, Luisa M Zintgraf, Joost van Amersfoort, Sebastian Farquhar, Nicholas Donald Lane, and Yarin Gal. "Prospect Pruning: Finding Trainable Weights at Initialization Using Meta-Gradients." ICLR 2022.

3. **Milad Alizadeh**, Arash Behboodi, Mart van Baalen, Christos Louizos,

Tijmen Blankevoort, and Max Welling. "Gradient ℓ_1 Regularization for Quantization Robustness." ICLR 2020.

4. Emilien Dupont*, Adam Goliński*, **Milad Alizadeh**, Yee Whye Teh, and Arnaud Doucet. "COIN: COmpression with Implicit Neural Representations." Neural Compression Workshop at ICLR 2021 (*Spotlight*)

5. Emilien Dupont*, Hrushikesh Loya*, **Milad Alizadeh**, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. "COIN++: neural compression across modalities." Transactions on Machine Learning Research 2022.

In the "Single Shot Structured Pruning Before Training" paper Joost van Amersfoort, Sebastian Farquhar, and I were equally-contributing authors. We all contributed to all aspects of the project including the core ideas, coding, and experimentation. This work was not published at a conference or journal but provided the foundation for the follow-up paper "Prospect Pruning".

In the "Prospect Pruning" paper I was lucky to additionally have Shyam Tailor and Luisa Zintgraf as collaborators. Luisa had a wealth of knowledge and expertise on Meta-Gradients from her work on Meta Reinforcement Learning and contributed to the core ideas of the project as well as the writing and rebuttal of the paper. Shyam gave me advice on minor details of the algorithm and was instrumental on the implementation side, specifically in scaling up experiments to ImageNet dataset and preventing memory exhaustion in the computation of the loss function. He has included this work in his PhD thesis ("Practical processing and acceleration of graph neural networks") with a focus on Graph Neural Networks (GNNs). His expansion of Prospect Pruning is the first work that attempts to apply pruning-at-initialization to GNNs.

The paper "Gradient ℓ_1 Regularization for Quantization Robustness" was done during my internship at Qualcomm AI Research at Amsterdam. Mart van Baalen was my supervisor. Amirhossein Habibian was my manager. Arash Behboodi and Christos Louizos contributed to the main ideas of the paper and without them the project would not be possible.

The original idea for the COIN and COIN++ projects came from Emilien Dupont and Adam Goliński. I joined the project to explore the possibility of compressing neural representations using quantization and pruning. As the project evolved, my

role in the project expanded and I contributed to all aspects of the project. The authors gave permission to mutually use text, figures, and tables from the original papers.

Chapter 2

Background

This chapter provides a brief background of the main topics that the rest of this thesis covers and also reviews some relevant existing works in the literature; it does not aim to be a comprehensive survey of these topics.

2.1 Pruning Deep Neural Networks

2.1.1 Motivation

Pruning is a popular compression technique that directly addresses the problem of neural networks having a large number of parameters by removing a substantial portion of them from the model. Although pruning has been getting more research attention in recent years with the need for more efficient neural networks, it has a long history in deep learning and was explored as early as 1988 (Mozer et al., 1988; LeCun et al., 1990a; Hassibi et al., 1993b). There have been various motivations for pruning since then.

The early motivation for pruning neural networks was avoiding *overfitting* rather than making models more efficient. Pruning can be interpreted as following Occam's razor principle, which dictates that simpler models should be preferred to complex ones. This is also related to the model-selection problem, where the objective is to choose one model among a set of candidate models. The Minimum Description Length (MDL) principle provides a Bayesian interpretation of this objective (Hinton

et al., 1993). Pruning in this setup can be seen as a type of regularization and, in some cases, has been shown to *increase* the model's performance on the test set. Kuhn et al. (2021) show that robustness against pruning can be an effective predictor of generalization of neural networks.

Pruning can potentially also make models more *interpretable*. As we will see shortly, many pruning methods work by estimating the saliency of parameters to decide what to prune. Pruning can therefore be directly linked to the interpretability of individual parameters. Additionally, a much smaller network is generally more interpretable. (Hamblin et al., 2022; Gilpin et al., 2019)

There have also been works linking pruning to *adversarial robustness*. Various works link the existence of adversarial inputs to the over-parametrization and capacity of the networks (Tran et al., 2018; Kaya et al., 2018). Jordao et al. (2021) and Ye et al. (2019) show that instead of regularizing for a robustness metric, pruning can work as a general defense mechanism against adversarial attacks.

However, as models have grown in size, taking advantage these potential pruning benefits have become more challenging. But the gains in terms of making neural networks efficient remain at hand and therefore storage and computation efficiency that comes with pruning has been the main focus of pruning research efforts. In recent years research has focused on pruning larger portion of parameters while affecting the performance of the model the least. Wang et al. (2019) for example show that they can prune 96% of parameters and 94% of FLOPS of a ResNet32 with only 2.25% drop in accuracy on CIFAR-10 dataset.

2.1.2 What to Prune?

The most important aspect of pruning neural networks is deciding which parameters to keep. Pruning methods generally fall into two categories based on how they make this decision: saliency-based methods and penalty-based methods.

The majority of pruning methods try to measure the importance of parameters and keep the top most-important parameters. In its simplest form the magnitude of a parameter can be used as a proxy for its importance and only parameters with the largest absolute weight are kept. Although magnitude pruning is very simple, it often delivers a competitive baseline in comparison to more sophisticated algorithms

(Gale et al., 2019). A more well-founded saliency criteria is the sensitivity of the loss function to removal of the parameter under consideration. If the loss function is not increased after pruning a parameter it makes sense to assume that removing it should not have an impact on the performance of the model.

However, there are problems with evaluating this metric in practice. A naive approach to evaluate this metric would be to remove each parameter individually, do a forward-pass to measure change in loss, and rank parameters based on how much the loss changed. However, this approach is not practical given that models have millions and billions of parameters. Moreover, this is only ideal if the objective is to remove a single parameter. In practice, we prune large number of parameters in a single step and this approach would not take the interaction between parameter into account.

A common approach to make this metric tractable is to build a linear or quadratic model approximation of the model and estimate the effect of pruning. This is done by using the Taylor expansion of the loss with respect to parameters to define a saliency score for each parameter:

$$\delta\mathcal{L} \approx \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\top}\delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^{\top}\mathbf{H}\,\delta\boldsymbol{\theta}, \tag{2.1}$$

where $\mathbf{H} = \nabla^2_{\boldsymbol{\theta}}\mathcal{L}$ is the Hessian matrix.

Using this approximation, pruning the k-th parameter would lead to the following change in the loss function:

$$\mathcal{L}\left(\boldsymbol{\theta} - \theta_k \mathbf{e}_k\right) - \mathcal{L}(\boldsymbol{\theta}) \approx -g_k\theta_k + \frac{1}{2}H_{kk}\theta_k^2, \tag{2.2}$$

where e_k is a one-hot vector with a single one at position k.

A lot of work in pruning literature deal with when and how to compute or estimate the terms in Equation 2.1. As we'll see in the next section pruning can happen at various stages of the training pipeline and that choice directly affects equation above. If pruning happens when the network has converged, the first-order term in the expansion is negligible, and hence these methods resort to using $\mathbf{H}$. If pruning happens at the beginning (or early stages) of training the first-order term could

still be dominant.

However, historically pruning has happened at the end and there's been a lot of focus on how to compute the second-order Hessian term. Hessian based approaches show better results compared to magnitude pruning, but the complexity of computing Hessian is $O(W^2)$ which makes it unfeasible for modern neural networks that have billions of parameters. There has been a lot of research on making Hessian computation more efficient. For example LeCun et al. (1990a) assume the Hessian matrix is diagonal, Singh et al. (2020) assumes block-wise diagonality, and Wang et al. (2019) use Kronecker factorization to find a basis in which the Hessian is in fact diagonal. In some cases the empirical Fisher Information Matrix (FIM) can be used as an approximation of the Hessian matrix which is easier to compute (Theis et al., 2018).

The second type of pruning methods by incorporating a penalty term in the training objective. It is common to add a data-independent penalty to the training objective from the ℓ_p family of norms. The ℓ_2 norm, also known as weight decay, is the most common regularizer but for the purpose of pruning, the ℓ_1 regularization, known as LASSO, is preferred as it is more likely to generate sparse solutions due to its geometric shape. In the limit as p approaches 0 we will have the ℓ_0 norm, also known counting norm, which is defined as the number of non-zero elements in the vector θ which is exactly the quantity we are interested in minimizing However, this term is non-differentiable and cannot be optimized using backpropagation. Louizos et al. (2017) propose using the continuous relaxation trick introduced by Maddison et al. (2017) to turn the ℓ_0 norm into a learnable parameter that can be optimized.

Dropout (Srivastava et al., 2014) is another type of regularization which works by stochastically pruning units during training. Molchanov et al. (2017) formulate dropout as an optimization problem by placing trainable gates in front of each parameter or channels and then pushing the probability of parameters getting pruned high during training. Gomez et al. (2019) apply dropout primarily to parameter with low importance in order to learn networks that learns to be explicitly robust against post-training pruning.

2.1.3 When to Prune?

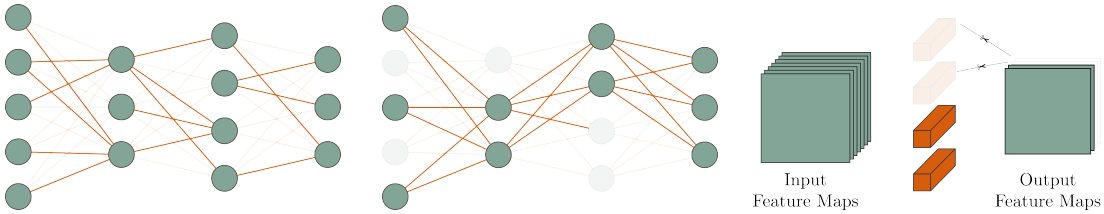
An important aspect of pruning, especially for saliency-based methods, is the pruning schedule. The most common and well-studied setup is to take a fully-trained neural network as input, prune it based on some saliency metric, and then potentially follow by some fine-tuning of the pruned model in order to recover the lost accuracy.

There is a common belief (which is also implicit in the popularity of this pruning schedule) that having an oversized neural network is crucial for successfully training accurate models and that training sparse models from scratch is unable to match their performance. However, a recent seminal work Frankle et al. (2019) shows that there exist sparse sub-networks within a neural network that have the potential to be trained from initialization and achieve comparable performance to the original model. The authors call these subnetworks "lottery tickets". Doing pruning at initialization (or at least early in training) has huge implications for improving the efficiency of training neural networks and more generally for a better understanding of neural network training. This has sparked a focus on pruning neural networks at initialization to find such *tickets*. This is also the focus of Chapter 3

Regardless of when pruning happens, it can happen in a single step or gradually. There have been many methods that explore pruning schedules, which show the effectiveness of pruning in multiple steps. Zhu et al. (2018) for example, uses an exponential schedule combined with simple magnitude criteria for deciding pruning targets at each pruning step and achieving very competitive results. Some methods prune models *iteratively* and allow pruned parameters to be reincarnated into the network. Iterative and incremental pruning allow for re-evaluation of saliencies, which can iron out the approximation inaccuracies.

2.1.4 Practical Considerations

An important aspect of pruning that directly affects how much they make neural networks more efficient on hardware platforms, is the granularity of the pruning. Pruning individual weights, known as *un*structured pruning, results in sparse weight matrices, but sparsity only leads to speed improvements with specialized hardware (Sze et al., 2017). Moreover, since sparse weights do not induce sparse



(a) Unstructured FC **(b)** Structured FC **(c)** Structured Conv

Figure 2.1: Structured vs. unstructured pruning. Unstructured pruning leads to sparse tensors which are not more efficient in hardware without special support. Structured pruning on the other hand simply leads to tensors with smaller dimensions.

activations, they do not reduce run-time memory footprint either. To gain significant inference speed-up by pruning on any standard compute device, hidden units or entire convolutional channels have to be removed (structured pruning) instead of individual weights. Figure 2.1 illustrated the difference between unstructured and structured pruning in fully-connected and convolutional layers.

Pruning has been historically used for making deployment of neural networks more efficient but with the recent advances pruning-at-initialization methods have made in the last few years, pruning has the potential to make training more efficient too. This is more important than ever with models getting large. For example, Strubell et al. (2019) estimated that training BERT (Devlin et al., 2019) on GPU causes the same carbon emission a flight across America.

Another important factor is the need for fine-tuning after or in-between pruning steps in many methods. Not only it is very costly in terms of additional training time but fine-tuning greatly affects the feasibility of pruning for practitioners. Not only it requires access to (often labeled) data but it also needs in-depth knowledge of the training pipeline. This is in contrasts with many post-training quantization methods that can work ad-hoc. Recent work shows that when a very simple pruning method like magnitude pruning is combined with carefully chosen fine-tuning and scheduling, the gains that come from adopting fancier methods begin to disappear (Laurent et al., 2020).

2.2 Quantizing Deep Neural Networks

2.2.1 Motivation

Neural networks are typically trained and stored in Single-precision floating-point format (FP32). Quantization is one of the most effective ways to compress and reduce their computation and storage costs. Unlike pruning which changes the architecture of networks in significant ways, quantization simply reduces the bit precision of weights and activations. In addition to immediate memory and storage savings that come as a result of using fewer bits, most quantization schemes also move from floating-point arithmetic domain to integer-only arithmetic, which can be implemented more efficiently on hardware, and is also more commonly available on low-end hardware platforms. There have been many successful examples where networks have been quantized to 8 or 4 bits with little effort and negligible performance degradation.

There have been many works that show successful quantization of neural networks to 8 or 4 bits with negligible drop in performance. When quantizing to 4 bits, the memory footprint of the network is reduced by a factor of 32 while the cost for performing MACs is reduced quadratically by a factor 64 (Nagel et al., 2021). Somewhat surprisingly, Courbariaux et al. (2015, 2016), Rastegari et al. (2016), and Liu et al. (2020) have shown continuous improvements in pushing quantization to the limit, i.e., representing values with a single bit, leading to Binary Neural Networks (BNNs) When both weights and activations are binary values, the MAC operation in neural networks is reduced to binary `XOR` and `POPCOUNT` operations that are fast, intrinsic operations supported on most hardware platforms. Rastegari et al. (2016) report 58X speed up on CPU training by using BNNs

In addition to being a go-to technique when deploying models, quantization has also been used to improve training as well, for instance, Hubara et al. (2017) and Zhou et al. (2016) use quantization in the backward pass to speed up training using low-precision gradients.

Unlike unstructured pruning which requires special hardware support to reap the benefits of model compression, quantization is very hardware-friendly. Most recent GPUs now support INT4, INT2, and INT1 (for 4, 2, and 1 bytes) support, and

more specialized platforms have been designed to work with even fewer bits.

2.2.2 Quantization Fundamentals

In this section, we introduce the most common types of quantizers used in practice and is illustrated in Figure 2.2. Quantization is often formulated as a round-to-nearest operation where real numbers from weights and activations are mapped to a grid of integers. The primary hyperparameters characterizing these quantizers are the number of available bits and the range covered by the grid.

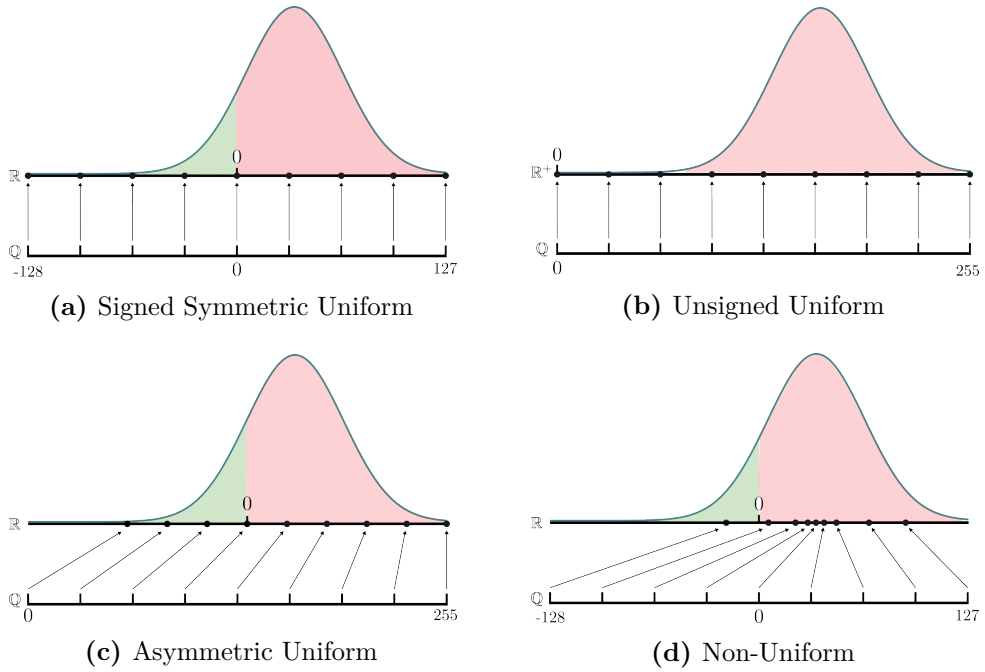
Before diving into the details of these quantizers, it is worth noting that while round-to-nearest is the predominant approach in quantization, it is not necessarily the only way or the best approach to map values to the quantization grid. Nagel et al. (2020), for example, stochastically map weights to the quantization grid and observe that there exist solutions that perform better than round-to-nearest quantizers. However, systematically finding these good solutions with no approximations is an NP-hard optimization problem.

Signed Uniform Quantization is the simplest quantization scheme where the quantization grid is placed symmetrically around zero, and real-valued zero is mapped to integer 0. Its hyperparameters are the number of bits b, and the scale factor s, which together determine the range of numbers covered by the quantization. The function that maps values to the grid is defined as:

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil ; \text{min} = -2^{b-1}, \text{max} = 2^{b-1} - 1\right) \tag{2.3}$$

$$\mathbf{x}_{\text{quant}} = s\,\mathbf{x}_{\text{int}} \tag{2.4}$$

Unsigned Uniform Quantization is a slightly modified version of the previous scheme that is useful for cases where the target numbers are non-negative. This frequently happens in neural networks, for instance, with ReLU non-linearity functions or when zero-padding in convolutional layers. In these scenarios, there is no point in wasting grid points for negative values that will never happen. To avoid that, the grid is shifted so that the very first integer is aligned with real-valued zero.



(a) Signed Symmetric Uniform **(b)** Unsigned Uniform

(c) Asymmetric Uniform **(d)** Non-Uniform

Figure 2.2: Common quantization schemes for compressing neural networks. Non-symmetric schemes make better use of the grid by aligning it with the desired range. In uniform quantizers, every integer level represents the same step size, while the non-uniform quantizer can assign more levels to the sub-range where values are more frequent.

The mapping is then given by:

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil ; \text{min} = 0, \text{max} = 2^b - 1\right) \tag{2.5}$$

Asymmetric Uniform Quantization takes the idea of shifting the grid in the previous scheme further to utilize discrete levels to the fullest. The shifting aligns the grid it with the min/max that we care about, resulting in grids that are asymmetric around zero. One thing we have to be careful about in doing so is making sure that zero is still represented on the quantized grid without introducing any errors. This is not ideal. This is important because many operations like ReLU or padding introduce actual zeros, and we would like to avoid accumulating errors. This is achieved by introducing zero-point z, which is an integer offset. The mapping is given by:

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; \text{min} = 0, \text{max} = 2^b - 1\right) \tag{2.6}$$

$$\mathbf{x}_{\text{quant}} = s\left(\mathbf{x}_{\text{int}} - z\right) \approx \mathbf{x} \tag{2.7}$$

Non-Uniform Quantization. Previous schemes make efficient use of the grid by shifting it, but a jump from one point on the grid to the next always represents the same delta on the real-valued axis. The next step in making more efficient use of the limited levels is to take the distribution of the numbers into account in addition to the range. We can assign more levels, and hence more precision, to intervals where numbers are more likely to occur, resulting in non-uniform schemes. While this scheme can lead to better performance, it introduces complexities in hardware implementations (Liu et al., 2022).

2.2.3 Post-Training Ad-hoc Quantization

Post-training Quantization (PTQ) takes a fully-trained neural network and directly quantizes it. Compared to pruning, which often requires fine-tuning and, therefore, access to the training pipeline, PTQ requirements are more relaxed.

While the range of weights is already known and fixed, we do need to decide the quantization range for activations. This can be done by doing forward passes using a few batches of data for calibration. The most straightforward approach is to track the Min/Max values observed during calibration and set the range to cover the whole observed range. The problem with this approach is that outliers can significantly affect the quantization performance and result in quantization levels that we almost never map values to. One way to alleviate the issue is to use Mean Squared Error (MSE) and set the quantization range such that the MSE between quantized and actual values are minimized (Nagel et al., 2021; Peters et al., 2018)

Nagel et al. (2021, 2019b) propose solutions for other common issues with PTQ, such as an imbalance in convolutional channel scales and dealing with high bias terms.

2.2.4 Learning Quantized Neural Networks

The PTQ approach introduced in the previous section works well for modest levels of quantization. However, when targeting lower bit widths such as 4, 2, or 1 bits, resorting to *learning* quantized weights often leads to better performance. This is

especially true for binary neural networks where PTQ often leads to models making random classifications.

The most common way to learn quantized values, also known as Quantization-Aware Training (QAT), works by placing quantizers in front of the FP32 weights as *operators* in the computation graph. Instead of learning quantized weights directly, we learn FP32 weights that expect to get quantized. When training is finished, the learned FP32 weights are passed through quantizers for one last time to get the final quantized values that replace real-valued weights. The main challenge in this formulation is that the rounding operation in quantizers is not differentiable and therefore, gradients cannot get backpropagated to the FP32 weights without applying some tricks. Bengio et al. (2013b) proposed Straight Through Estimator (STE) approximation to deal with this problem:

$$\frac{\partial \lfloor x \rceil}{\partial x} = 1. \tag{2.8}$$

STE ignores the rounding operation and passes gradients backward as if the rounding was not part of the computation graph. While this sounds counter-intuitive and is clearly a biased estimator, in practice, this has led to significantly good results, albeit with the help of additional modifications and tricks that we will cover in Chapter 4.

2.2.5 Practical Considerations

An important design decision in both PTQ and QAT is the *granularity* of quantization. While the same quantizer can be used for the entire weight tensor, using a different quantizer per channel in convolutional layers can often improve the performance, albeit at the cost of adding an additional FP32 scale value per channel. Moreover, quantizing different layers to different bit widths can lead to significant savings but adds many more hyperparameters that need to be tuned (Cai et al., 2020; Micikevicius et al., 2017; Dong et al., 2019).

BatchNorm layers often need special attention in quantized neural networks. When deploying trained neural networks, it is common to fuse BatchNorm scale values into the adjacent weights tensor. BN fusing avoids extra data movement and computation but can significantly change the statistics of weights and channels

and therefore has implications for the quantizers that are tuned for the original weight tensor. Therefore it is essential to simulate this fusing operation when doing quantization learning (Krishnamoorthi, 2018a; Jacob et al., 2018).

Fusing also applies to many non-linearity functions, such as ReLU. More and more hardware platforms have fused operations for MACs followed by non-linearity, so it is essential to make sure quantization happens *after* fused non-linearity.

2.3 Meta-Gradients in Neural Networks

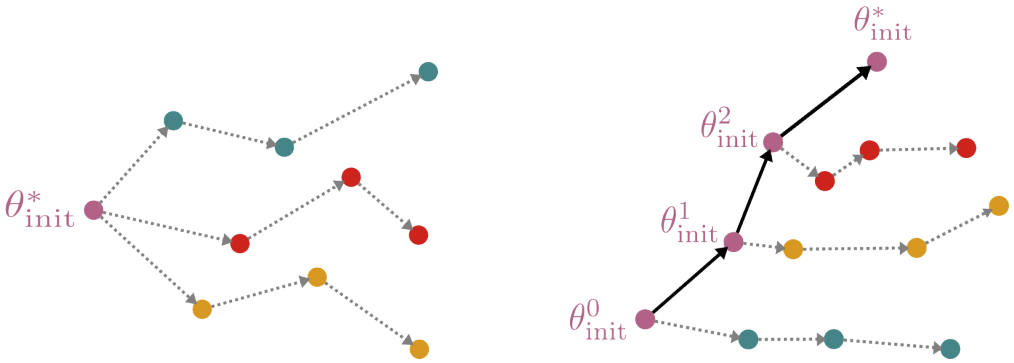
The subsequent chapters in this thesis explore various aspects of compression in deep learning ranging from quantization and pruning to data compression, but the common thread in all of them is the use of higher-order gradients, also known as meta-gradients. This section introduces meta-gradients and some of their applications outside compression.

The main idea of meta-gradients is that gradients computed during backpropagation can themselves be part of another objective and therefore, another computation graph. Given that these gradients are differentiable, one could train the parameters of the network in the direction of this meta-objective by computing gradients-of-gradients.

2.3.1 Gradient-Based Meta-Learning

A performant neural network requires training with a large number of training samples. On the other hand, humans can learn from demonstration with a few examples. Meta-learning, or learning-to-learn, aims to bring the same capability to the neural networks.

A family of methods, popularized by MAML (Finn et al., 2017a; Zintgraf et al., 2019b; Antoniou et al., 2019), use meta-gradients to achieve this goal. Our desire is to find models that can be fine-tuned to perform well on new unseen tasks using just a few gradient steps. MAML formulates this goal as an optimization problem consisting of two nested loops. The inner loop mimics what the network faces at test time, i.e., it starts from an initialization point and makes a few gradients step towards adopting to the given task. However, the updates to parameters are only



(a) Test time **(b)** Training time

Figure 2.3: Meta-learning an initialization that can be quickly fine-tuned to new tasks at test time. Dashed arrows show the directions given by normal gradients while bold arrows represent directions given by the meta-gradients. Different colors represent different tasks.

temporary and are only used by the outer loop. The outer loop moves parameters in the direction we care about, i.e., it updates the initialization that the inner loop started from, such that inner loop performance is improved. This direction is given by the meta-gradients of the loss at the end of the inner loop with respect to parameters at the beginning of the loop. The outer loop gets a new task every time, and by repeating the outer loop for many epochs, one would hope to get to a good initialization for the network. Figure 2.3 summarizes the process.

In Chapter 3 we will use a similar approach as a pruning criterion.

2.3.2 Meta-Gradients for Regularization

Another use case of meta-gradients is as a penalty term for regularizing neural networks. An example of this is in GANs. GANs are notoriously difficult to train and can suffer from mode collapse, training divergence, and other issues. Arjovsky et al. (2017) proposed enforcing smoothness of the discriminator function using 1-Lipschitz constraint to avoid the caveats in the training of GANs. A naive way to achieve this is by clipping the weights, but it introduces other issues, such as vanishing gradients and slowing down training. (Gulrajani et al., 2017) proposed using meta-gradients instead. A differentiable function is 1-Lipschitz if and only if it has gradients with ℓ_2-norm of at most 1 everywhere hence the authors penalize the ℓ_2-norm of the gradient of the critic with respect to its input.

In Chapter 4 we will use a similar approach in the context of quantization.

2.3.3 Meta-Gradients for Hyperparameter Optimization

Another application of meta-gradients is tuning hyperparameters. When dealing with a handful of parameters that need tuning, simple methods like random or grid search often suffice. When there are more parameters to tune, more scalable solutions such as Bayesian Optimization (Kandasamy et al., 2020; Močkus, 1975) can be effective. However, these methods often fail in high dimensions.

Maclaurin et al. (2015) propose using meta-gradients instead to learn hyperparameters in a loop that optimizes validation loss directly. The authors show that meta-gradients allow optimization of the validation loss with respect to thousands of hyperparameters, including fine-grained learning rate schedules, per-layer initialization distributions of weights, per-input regularization schemes, and per-pixel data preprocessing. They further show that training data can be viewed as just another hyperparameter and meta-learn synthetic MNIST training examples.

Lorraine et al. (2020) further propose using Hessian approximation algorithms in order to scale up hyperparameter tuning with meta-gradients to millions of parameters. They jointly tune weights and hyperparameters and learn a data-augmentation network where every weight is a hyperparameter tuned for validation performance.

2.4 Neural Data Compression and Implicit Neural Representation

The amount of data that the world is producing is growing rapidly. Data compression techniques, which aim to represent information using fewer bits than the original signal, are therefore more important than ever. For example Evans et al. (2020) forecast that video streaming will account for 82% of all internet traffic and 1% of global carbon emissions by 2022, and therefore even small improvements in compression methods can lead to massive savings.

Neural data compression is the application of neural networks in achieving this

objective. Unlike classic compression methods that are designed using deep domain knowledge and hand-crafted features, neural compression automatically constructs compression codecs from raw training data. While there were attempts at using neural networks for image compression in the late 1980s (Sonehara, 1989), recent interest and excitement in neural compression were prompted by the success of generative models. The bottleneck architecture in VAEs, for instance, forces an encoder to learn a compressed representation of the signal and can be used for data compression

In addition to common data types such as images and video, neural-based compression can be especially impactful for new or domain-specific data types, such as scientific data or VR content, where developing custom codecs could be time-consuming and expensive.

Another closely related topic to neural networks and data representations is Implicit Neural Representation, typically called INR. INR is a way to parameterize signals, such as images, shapes, videos, audio, 3D scenes, and many more. They are fundamentally different from conventional signal representations that are typically discrete. Images for instance are often represented as a grid of pixel with discrete RGB values and audio signals are represented by sampling a sequence of amplitude values. The true underlying functions that parametrize signals is untractable and the main idea behind INRs is to approximate these functions via neural networks. Having continuous functions representing signals has immediate applications for example super-resolution.

Conventional neural network architectures fail to parametrize natural signals. One reason is that these networks are not shift-variant, i.e. they have a hard time learning the same function for different input coordinates, which is common in many natural signals. Another reason is ReLU networks are piecewise linear and hence the second derivative is zero everywhere. However many natural signals, such as waves, have second order information. Sitzmann et al. (2020b) propose using periodic non-linearities instead of ReLUs to solve both problems.

Chapter 3

Pruning at Initialization Using Meta-Gradients

In this chapter, we explore the application of meta-gradients in neural network compression as an informative signal for the problem of pruning-at-initialization. We are specifically motivated by structured pruning because it can significantly reduce training time. We begin by extending standard pruning-at-initialization methods to the structured pruning scenario and evaluate their effectiveness. We also evaluate how explicitly biasing pruning decisions towards removing more FLOPS affect the performance.

We then identify a fundamental limitation in the formulation of current standard methods due to ignoring *model trainability* when computing saliency scores. We show how meta-gradients can be used as a more informative signal to find better trainable subnetworks at initialization.

3.1 Background

Frankle et al. (2019) provided empirical evidence for the existence of sparse subnetworks within fully trained neural networks that could have been trained from initialization and achieve accuracies comparable to the original network. This is in contrast to the conventional belief that overparameterized models are essential for

successful training. These "winning tickets" were originally found in an iterative process where, in each iteration, the network is trained to full convergence, followed by pruning a subset of the weights by magnitude. The values of the remaining weights are then rewound to their value at initialization, and the process is repeated iteratively until the desired sparsity level is achieved.

This process, known as Lottery Ticket Rewinding (LTR), is extremely compute-intensive and prone to failures. For instance, Frankle et al. (2020) achieve better results by rewinding weights not all the way back to initialization but to the early stages of training instead. LTR is especially prone to failure for more difficult problems (e.g., training on ImageNet), where weights are changed to their state at several epochs into the training.

Pruning at initialization is desirable because the benefits of pruning (in terms of memory and speed) can be reaped during training rather than only at inference/deployment time. A recent line of work proposes alternative practical solutions to identify these subnetworks *before* any training happens and without the cost of retraining the network iteratively (Lee et al., 2019d; Wang et al., 2020a; Jorge et al., 2021; Tanaka et al., 2020). This class of methods uses gradients to assess the importance of neural network weights. These gradients are sometimes known as Synaptic Saliencies and are used to estimate the effect of pruning a *single* parameter on various objectives, typically the loss function.

We introduced the main concepts in pruning in §2.1. In the rest of this chapter we review the concepts that pruning-at-initialization builds upon in more details.

Definitions. Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) : i = 1, \dots, N\}$ be the training dataset for a supervised learning task. Denote by $f_{\boldsymbol{\theta}}$ a neural network with parameters $\boldsymbol{\theta}$, and let $\mathcal{L}(f_{\boldsymbol{\theta}}, y)$ be the neural network's loss for data point $(\mathbf{x}, y)$. Let $k < 1$ be the target prune ratio: the amount by which we hope to reduce the network's size.

Saliencies Scores. Classic post-training pruning methods typically use the Taylor expansion of the loss with respect to parameters to define a saliency score for each parameter: $\delta\mathcal{L} \approx \nabla_{\boldsymbol{\theta}}\mathcal{L}^\top \delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^\top \mathbf{H}\, \delta\boldsymbol{\theta}$, where $\mathbf{H} = \nabla^2_{\boldsymbol{\theta}}\mathcal{L}$ is the Hessian matrix (LeCun et al., 1990b; Hassibi et al., 1993a). When the network has converged, the first-order term in the expansion is negligible and hence these methods resort to

using $\mathbf{H}$.

In their seminal work SNIP, Lee et al. (2019d) showed that the same objective of minimizing the change in loss could be used at initialization to obtain a trainable pruned network. At initialization, the first-order gradients $\nabla_{\boldsymbol{\theta}}$ in the local quadratic approximation are still significant, so the higher-order terms can be ignored. Therefore, the computation of the parameter saliencies can be done using simple backpropagation.

The Taylor expansion approximates the effect of small *additive* perturbations of weights on the loss. To better approximate the effect of *removing* a weight, Lee et al. (2019d) modify the computation graph and attach a *multiplicative* mask, initialized to 1, to each weight. This does not change the forward-pass of the network, but it enables us to form the Taylor expansion around the mask values, rather than the weights, to estimate the effect of changing the mask values from 1 to 0. More specifically, SNIP computes the saliency scores according to:

$$s_j = \frac{|g_j(\mathbf{w}, \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}, \mathcal{D})|} \ , \tag{3.1}$$

with

$$g_j(\mathbf{w}, \mathcal{D}) = \frac{\partial \mathcal{L}(\mathbf{c} \odot \mathbf{w}, \mathcal{D})}{\partial c_j} \ , \tag{3.2}$$

where m is the number of weights in the network, $\mathbf{c} \in \{0, 1\}^m$ is the pruning mask (initialized to $\mathbf{1}$), $\mathcal{D}$ is the training dataset, $\mathbf{w}$ are the neural network weights, $\mathcal{L}$ is the loss function, and $\odot$ is the Hadamard product.

These saliency scores are computed at initialization using one (or a few) mini-batches from the training set. The global top-k weights with the highest saliency scores are retained ($c_j = 1$), and all other weights are pruned ($c_j = 0$), before the network is trained.

Our methods in the next section rely on computing saliency scores in a similar fashion to Equation 3.2 but it is worth noting that there are alternative ways to use gradients for saliency estimation. Instead of keeping change in loss minimum, GraSP proposes an alternative criterion for pruning at initialization based on preserving gradient flow (Wang et al., 2020a). GraSP keeps weights that contribute most to the norm of the gradient. While this method works well in unstructured pruning,

especially at high sparsity levels, we show in §3.4.1 that GraSP's approximation has shortcomings that are particularly problematic for structured pruning. We therefore use Equation 3.2 in the rest of this chapter as the basis for computing saliencies.

3.2 Structured Pruning Using Gradients

While the criterion in Equation 3.2 works relatively well in practice, it is important to explicitly note some of the assumptions made in this approximation before extending it to the structured scenario:

- We use a continuous relaxation of masks to allow differentiation and back-propagation instead of solving the discrete program.
- We do not have access to $\mathcal{L}$ on the true data distribution and therefore approximate it with an MC estimator. In our case, this is a single batch of data at each pruning step.
- The saliency scores estimate the effect of pruning a single parameter in isolation, that removing one set of weights will not change the saliency of another set of weights. However, we typically prune many parameters at a time. Post-train methods use the Hessian to account for correlations between parameters but computing this second-order term is difficult for large tensors.

Each assumption above also applies to structured pruning, but in the structured setting these assumptions are significantly stronger, requiring additional justifications. Unstructured models have a mask variable with potentially millions of entries (the number of weights), while structurally pruned VGG-19 will have roughly 5,000 entries. Removing a single (output) channel can, for example, lead to the removal of 512×3×3 weights, such as in the later layers of VGG and ResNet (Simonyan et al., 2015; He et al., 2016a). This means that the interaction between any two entries will often be much larger, making the second and third assumptions less plausible.

Extending any unstructured method to a structured method is therefore not trivial, especially when the original score is as noisy as in pruning-at-initialization. This is just one reason that moving to structured pruning is highly non-trivial and one of

our motivations for looking beyond gradients and using meta-gradients in §3.3

3.2.1 Extending Gradient Saliency to Structured Pruning

SNIP defines $\mathbf{c}$ with the same shape as the weights, allowing it to turn off individual weights. We instead define $\tilde{\mathbf{c}}$ to remove entire operations. Unlike unstructured pruning-at-initialization we want to find the structured binary mask tensor $\tilde{\mathbf{c}}$ which minimizes the change in loss:

$$\begin{aligned} &\arg\min_{\tilde{\mathbf{c}}} \left| \mathcal{L}\Big(f_{\boldsymbol{\theta}\odot\tilde{\mathbf{c}}}(\mathbf{x}), y\Big) - \mathcal{L}\Big(f_{\boldsymbol{\theta}}(\mathbf{x}), y\Big) \right| \\ &\text{s.t.} \quad \tilde{\mathbf{c}} \in \{0,1\}^p \text{ and } \| \tilde{\mathbf{c}} \|_0 = k \times | \tilde{\mathbf{c}} |, \end{aligned} \tag{3.3}$$

where p denotes the number of convolutional channels and hidden units in fully-connected layers, and $f_{\theta\odot\tilde{\mathbf{c}}}$ is the model obtained by applying the structured mask $\tilde{\mathbf{c}}$ to the unpruned model. The requirements specify that the mask $\tilde{\mathbf{c}}$ has to be binary with the ratio of number of zeroes to ones of k. This discrete optimization problem is intractable. We use the first-order estimates in Equation 3.2 to approximate the optimization problem and compute individual saliencies for each element in $\tilde{\mathbf{c}}$. While $\mathbf{c}$ has one entry for every weight in the model, $\tilde{\mathbf{c}}$ is its structured counterpart with many fewer entries. Specifically, for convolutional layers, an output channel gets a single binary mask variable governing its entire spatial extent. Similarly, fully-connected layers have a binary mask per hidden unit; we visualize this in Figure 3.1.

In unstructured pruning, once the saliency scores and hence the pruning mask are computed, pruning can be applied by element-wise multiplication of the mask and weights. In structured pruning however, we can remove channels and units from the weight tensor leading to a faster model. Masking, therefore, can be implemented by changing the shape of the weight tensors and is equivalent to using a smaller model. Structured pruning also changes the activation dimension, which means we need to *remove* the corresponding channel/hidden unit in the subsequent layer, further speeding up the model.

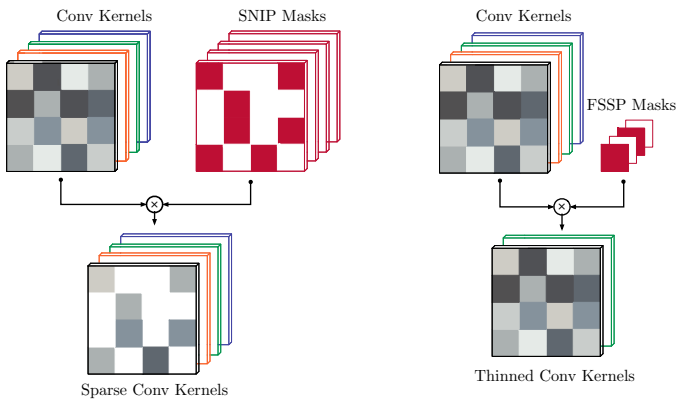
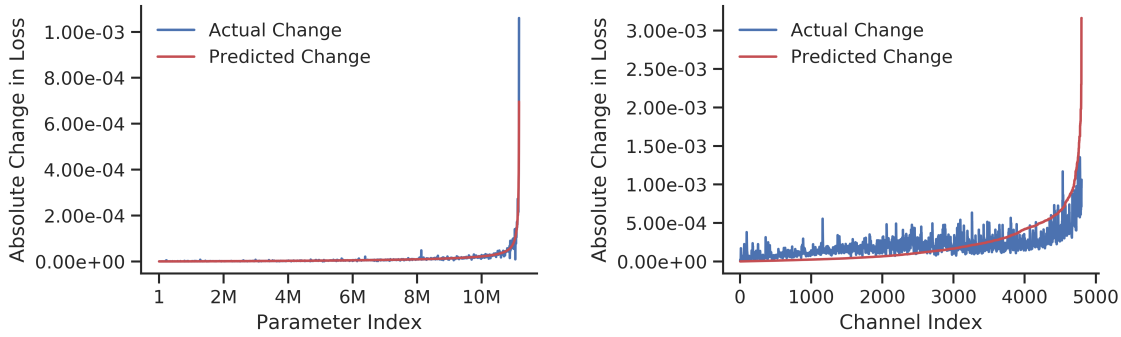


Figure 3.1: Binary mask tensors for a convolution weight kernel in SNIP vs. FSSP. There is one mask per weight in the unstructured pruning while in the structured case, the binary mask $\tilde{\mathbf{c}}$ has one entry per convolutional *channel*.

Evaluating Objectives for Structured Pruning. Given that the assumptions in gradient saliency approximation are stronger in structured pruning, we do a preliminary investigation to see if extending saliencies to structured pruning is sensible. The saliency scores are predictions of change in loss induced by pruning. In Figure 3.2, we assess how accurate these predictions are in practice for predicting the change in the loss. We do this by removing individual weights, performing a forward pass, and measuring the actual change in the loss. We show on the left that the unstructured SNIP approximation is close to the actual change in the loss when a single weight is removed. On the right, we show that there is significantly more noise in the structured case, but the predicted loss change correlates strongly with the actual change.

We evaluate the alternative GraSP objective in more detail in §3.4.1 and show that its approximation does not extend to a structured setting.

Dealing with Residual Connections. ResNets (He et al., 2016a) impose special architectural requirements for structured pruning. A ResNet is structured in residual blocks that take some input x, do operations on this input to obtain $f(x)$ and then add the original input to the result $f(x)+x$ (the "skip connection"). When pruning, it is important that the final operation of $f(\cdot)$ recovers the original shape of x such that $f(x)+x$ can be computed. Some blocks in a ResNet downsample the spatial dimension. In that case, the identity connection is also parametrized: $f(x)+g(x)$,



(a) Unstructured **(b)** Structured

Figure 3.2: Accuracy of the approximation in SNIP's objective in unstructured and structured pruning with the SNIP objective on VGG-19. We compare the predicted change in loss from the Taylor expansion (scores) with the actual change in loss (normalized). The X-axis is sorted by the scores. While scores in structured pruning are noisy, we can still successfully identify parameters that contribute the most to the change in the loss. For the unstructured curve, we compute the actual change in loss for every five thousandth weight to improve computation efficiency.

where both $f(\cdot)$ and $g(\cdot)$ are strided convolutions of size 3 and 1 respectively.

When pruning a ResNet we need to make sure the output dimensions continue to match. In the case of a residual block with down-sampling, we share a mask parameter between the last convolution inside $f(\cdot)$ and $g(\cdot)$. This means we have one score for removing a channel from *both* layers; see the visualization in Figure 3.3. When the residual block does not downsample, we link the mask of the last convolution inside $f(\cdot)$ back to the last convolution of the previous block.

This gradient binding approach is more principled than previous works like Wang et al. (2019) which prune layers linked by a skip connection separately and take the maximum score over tied channels.

Rescaling Initialization. In addition to the pruning objectives introduced earlier we need to think about the interaction of pruning and initialization scheme. Pruning at initialization changes the variance of activations at initialization and in the structured case even changes the number of activations. Previous research into initialization schemes for neural networks (He et al., 2015) shows the importance of the variance and the number of activations for model training. By pruning a model, we reduce the number of output channels of most layers, which means the

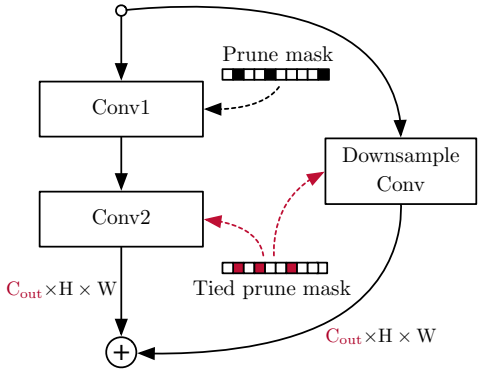


Figure 3.3: Gradient binding in ResNets's BasicBlock. Conv2 and Downsample Conv must have the same number of output channels such that their outputs can be added. We achieve this by using a *single* mask variable for both effectively binding the parameters.

variance no longer has the right value relative to the number of output activations. We attempt to correct for this by studying the effects of rescaling weights by the ratio: $\sqrt{\text{original fan out}/\text{pruned fan out}}$ which amounts to recovering the variance scaling suggested by He et al. (2015). However, we note that because large and small weights are not pruned uniformly, the resulting variance of the model weights of the pruned model is not exactly what one would get from initializing the model from scratch. The authors of SNIP reinitialize their models after pruning, which may address the same issue.

3.2.2 Compute-Aware Pruning

Different layers in a model have a different computational cost. For example the computational cost (in FLOPs) of a convolution operation is $2 \cdot H \cdot W \cdot C^{\text{out}} \cdot C^{\text{in}} \cdot K^2$ where H and W, are the height and width of the output, C_{out}, C_{in} are the number of out and in channels, and K is the size of the kernel. In earlier layers of the model, the spatial dimensions tend to be high, while in later layers the number of input channels increase. Since our aim is to remove as much compute from the model while preserving model accuracy, we extend the gradient saliency derived in the previous section to be compute-aware, by dividing the score by the normalized compute cost c_i per channel:

$$\bar{c}_i = \frac{c_i}{\sum_j c_j} \qquad \text{where} \qquad c_i \coloneqq 2 \cdot H_i \cdot W_i \cdot C_i^{\text{in}} \cdot K_i^2. \tag{3.4}$$

We calculate a retention score, $r_{i,j}$, for layer i, channel j, corresponding to each mask entry, which measures the impact on the loss per unit of compute removed:

$$r_i = \frac{\mathcal{S}_{i,j}}{\tilde{c}_i}, \tag{3.5}$$

where $\mathcal{S}$ was the original saliency tensor. This retention score can be high either if a channel has a big effect on the loss, or if the channel has a negligible effect on the compute cost. A low retention score means that the channel is either harmful to prune or would offer little speed-up. In practice, we would like to trade off the importance of compute and change in the loss, so we introduce a Laplace smoothing parameter λ (Manning et al., 2009): $\tilde{c}_i = \frac{\bar{c}_i + \lambda}{\sum_j \bar{c}_j + \lambda}$. A larger value of λ makes the pruning depend more on the predicted change in loss and pay less attention to the compute costs of different layers.

3.2.3 Single-Shot Structured Pruning

In this section we introduce our first method for structured pruning at initialization. Our method, Single Shot Structured Pruning (3SP), is easy to implement and has few hyperparameters to tune. The pruned model trains 2x faster (on a GPU) and performs inference 3x faster (on a CPU), with only a 0.5% loss in accuracy on CIFAR-10. Using the compute-aware variant we are able to increase compute reduction from 60% to 85%.

Speeding up the training of large neural networks is useful when the training data changes quickly and models need to adjust rapidly. One example of this is active learning (Settles, 2010), where trained models are used to identify the most informative datapoints to label for inclusion in the training data. In §3.4.1 we show how 3SP can be used to identify valuable data to acquire faster than a full model, allowing us to achieve better accuracy within a time-budget than an un-pruned model could.

We provide a step by step description of how 3SP and its compute-aware extension works in Algorithm 1.

Algorithm 2 Structured Compute-Aware Pruning - 3SP

Data: minibatch of data from dataset $\mathcal{D}$.
Result: Best model mask $\tilde{\mathbf{c}}^*$ for pruned model.
Initialization: NN with in channels C_l^{in}, spatial height and width H_l and W_l and kernel size K_l for each layer l, initialized with He et al. (2016a); $\tilde{\mathbf{c}}$ initialized at 1 for every entry; target pruning ratio p; compute smoothing term λ; $\boldsymbol{R} \in \mathcal{R}^{|\tilde{\mathbf{c}}|}$;

$g \leftarrow \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{c}}}$ ▷ 1^{st} order apprx. $\mathcal{S}$ using one minibatch.
if Compute-Aware **then**
  $c_l \leftarrow C_{\text{in}} \cdot H_l \cdot W_l \cdot K_l^2$ ▷ Calculate cost-per-layer.
  $\bar{c}_l \leftarrow \frac{c_l + \lambda}{\sum_j (c_j + \lambda)}$ ▷ Apply compute cost smoothing.
  $\bar{c}_l \leftarrow \frac{\bar{c}_l}{\max_j(\bar{c}_j)}$ ▷ Normalize compute cost.
  for Score g_i and associated layer l **do**
    $\boldsymbol{R}_i \leftarrow \frac{|g_i|}{\bar{c}_l}$ ▷ Convert 3SP score to cost-space
  end for
else
  $\boldsymbol{R} \leftarrow |g_i|$ ▷ Non-compute-aware 3SP score.
end if
$\boldsymbol{R}_{\text{threshold}} \leftarrow p$'th percentile of entries in $\boldsymbol{R}$
$\tilde{\mathbf{c}}^* \leftarrow \boldsymbol{R} \geq \boldsymbol{R}_{\text{threshold}}$ ▷ Keep channels/units with high score.

---

for $i = 1, \ldots, N$ **do** ▷ Train pruned model
  $\hat{\mathbf{w}}_{i+1} = \hat{\mathbf{w}}_i - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{w}}_i, \mathcal{D})$
end for

3.3 Training-Aware Pruning Using Meta-Gradients

While adopting pruning-at-initialization to structured domain perform relatively well, the degradation in accuracy is still significant compared to training the full model and LTR, making these methods impractical for many real-world problems (Frankle et al., 2021). In this section, we introduce the main contribution of this chapter. We identify a fundamental limitation in the objective formulation of current methods, namely that saliency criteria do not take into account the fact that the model is going to be trained after the pruning step. If our aim was to simply prune a subset of weights without affecting the loss, then these saliency

criteria are estimating the correct objective. However, this estimate does not take into account that we are going to train the weights after we prune them. We need a metric that captures the *trainability* of the weights during the optimization steps, rather than a single myopic estimate.

Many methods attempt to overcome this by pruning gradually and/or adding training steps between iterative pruning steps (Zhu et al., 2018; You et al., 2020; Jorge et al., 2021). Although this approach has been shown to be effective, it is expensive and cumbersome in practice and ultimately is an indirect approximation to the *trainability* criteria we are looking to incorporate into our objective.

In this section we introduce our method, Prospect Pruning (ProsPr). We note that for the problem of pruning-at-initialization, the pruning step is immediately followed by training. Therefore, pruning should take into account the *trainability* of a weight, instead of only its immediate impact on the loss before training. In other words, we want to be able to identify weights that are not only important at initialization, but which may be useful for reducing the loss *during training*. To this end, we propose to estimate the effect of pruning on the loss over *several steps of gradient descent* at the beginning of training, rather than the changes in loss at initialization.

More specifically, ProsPr models how training would happen by performing multiple (M) iterations of backpropagation and weight updates—like during normal training. We can then backpropagate through the entire computation graph, from the loss several steps into training, back to the original mask, since the gradient descent procedure is itself differentiable. Once the pruning mask is computed, we rewind the weights back to their values at initialization and train the pruned network. The gradient-of-gradients is also known as the *meta-gradient*. This algorithm is illustrated in Figure 3.4.

The higher-order information in the meta-gradient includes interactions between the weights during training. When pruning at initialization, our ultimate goal is to pick a pruned model, A, which is more trainable than an alternative pruned model B. That means we want the loss $\mathcal{L}(\hat{y}_A, y)$ to be lower than $\mathcal{L}(\hat{y}_B, y)$ at convergence (for a fixed pruning ratio). Finding the optimal pruning mask is generally infeasible since the training horizon is long (i.e., evaluation is costly) and the space of possible pruning masks is large. Unlike other methods that must compute the saliency scores

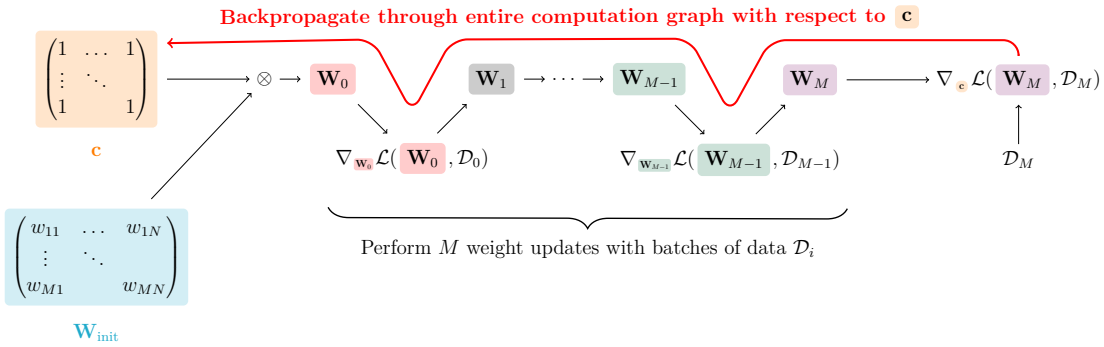


Figure 3.4: Visualization of computing saliency scores in our method, ProsPr. By backpropagating through several gradient steps we capture higher-order information about the objective that we care about in practice, i.e., saliency of parameters *during training* and not just at initialization.

iteratively, we can use the meta-gradients to compute the pruning mask in *one shot*. This picks a line in loss-space, which more closely predicts the eventual actual loss. This is because it smooths out over more steps, and takes into account interactions between weights in the training dynamics. Crucially, in the limit of large M, the match to the ultimate objective is exact.

In summary, our contributions are:

- We identify a key limitation in prior saliency criteria for pruning neural networks—namely that they do not explicitly incorporate trainability-after-pruning into their criteria.
- We propose a new pruning-at-init method, **ProsPr**, that uses meta-gradients over the first few training steps to bridge the gap between pruning and training.
- We show empirically that ProsPr achieves higher accuracy compared to existing pruning-at-init methods. Unlike other methods, our approach is *single shot* in the sense that the pruning is applied to the network initial weights in a single step.

3.3.1 Saliency Scores via Meta-Gradients

We now introduce ProsPr formally. After initializing the network weights randomly to obtain $\mathbf{w}_{\text{init}}$, we apply a weight mask to the initial weights,

$$\mathbf{w}_0 = \mathbf{c} \odot \mathbf{w}_{\text{init}}. \tag{3.6}$$

This weight mask contains only ones, $\mathbf{c} = \mathbf{1}$, as in SNIP (Lee et al., 2019d), and represents the connectivity of the corresponding weights.

We then sample $M+1$ batches of data $\mathcal{D}_i \sim \mathcal{D}^{\text{train}}$ ($i \in \{0, \ldots, M\}$; $M \geq 1$) for the pruning step, and perform M weight updates[1],

$$\mathbf{w}_1 = \mathbf{w}_0 - \alpha \nabla_{\mathbf{w}_0} \mathcal{L}(\mathbf{w}_0, \mathcal{D}_0) \tag{3.7}$$

$$\vdots$$

$$\mathbf{w}_M = \mathbf{w}_{M-1} - \alpha \nabla_{\mathbf{w}_{M-1}} \mathcal{L}(\mathbf{w}_{M-1}, \mathcal{D}_{M-1}). \tag{3.8}$$

Then, we compute a meta-gradient that backpropagates through these updates. Specifically, we compute the gradient of the final loss w.r.t. the initial mask,

$$\nabla_{\mathbf{c}} \; \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M). \tag{3.9}$$

Using the chain rule, we can write out the form of the meta-gradient beginning from the last step:

$$\nabla_{\mathbf{c}} \mathcal{L}(\mathbf{w}_M, \mathcal{D}) = \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D})(\nabla_{\mathbf{c}} \mathbf{w}_M), \tag{3.10}$$

repeating for each step until we reach the zero'th step whose gradient is trivial,

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D})(\nabla_{\mathbf{w}_{M-1}} \mathbf{w}_M) \ldots (\nabla_{\mathbf{w}_0} \mathbf{w}_1)(\nabla_c \mathbf{w}_0) \tag{3.11}$$

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D})(\nabla_{\mathbf{w}_{M-1}} \mathbf{w}_M) \ldots (\nabla_{\mathbf{w}_0} \mathbf{w}_1)(\nabla_c (\mathbf{c} \odot \mathbf{w}_{\text{init}})) \tag{3.12}$$

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}) \left[\prod_{m=1}^{M} (\nabla_{\mathbf{w}_{m-1}} \mathbf{w}_m) \right] \mathbf{w}_{\text{init}}. \tag{3.13}$$

[1]We formalize the weight updates using vanilla SGD here; in practice these may be different when using approaches such as momentum or BatchNorm (Ioffe et al., 2015). Since our implementation relies on automatic differentiation in PyTorch (Paszke et al., 2019a), we can use any type of update, as long as it is differentiable w.r.t. the initial mask $\mathbf{c}$.

In practice, we can compute the meta-gradients by relying on automatic differentiation software such as PyTorch (Paszke et al., 2019a). However, care must be taken to ensure that weights at each step are kept in memory so that the entire computation graph, including gradients, is visible to the automatic differentiation software. The saliency scores are now given by

$$s_j = \frac{|g_j(\mathbf{w}, \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}, \mathcal{D})|}, \tag{3.14}$$

with

$$g_j(\mathbf{w}, \mathcal{D}) = \frac{\partial \mathcal{L}(\mathbf{w}_M, \mathcal{D})}{\partial c_j}, \tag{3.15}$$

where $\mathbf{w}_M$ is a function of $\mathbf{c}$. Equation (3.15) stands in contrast to SNIP, where the saliency is computed using the loss at $\mathbf{c} \cdot \mathbf{w}_{\text{init}}$ rather than $\mathbf{w}_M$. The saliency scores are then used to prune the *initial* weights $\mathbf{w}_{\text{init}}$: the ones with the highest saliency scores are retained ($c_j = 1$), and all other weights are pruned ($c_j = 0$). Finally, the network is trained with the pruned weights $\hat{\mathbf{w}}_{\text{init}}$.

Algorithm 3 summarizes the proposed method.

3.3.2 First-Order Approximation

Taking the meta-gradient through many model updates (Equation (3.9)) can be memory intensive: in the forward pass, all gradients of the individual update steps need to be retained in memory to then be able to backpropagate all the way to the initial mask. However, we only need to perform a few steps[2] at the beginning of training so in practice we can perform the pruning step on CPU which usually has access to more memory compared to a GPU. We apply this approach in our own experiments, with overheads of around 30 seconds being observed for the pruning step.

Alternatively, when the number of training steps needs to be large we can use the following first-order approximation. Using Equation (3.13), the meta-gradient

[2]We use 3 steps for experiments on CIFAR-10, CIFAR-100 and TinyImageNet datasets

Algorithm 3 ProsPr Pseudo-Code

Inputs: a training dataset $\mathcal{D}^{\text{train}}$, number of initial training steps M, number of main training steps N ($M \ll N$), learning rate α
Initialise: network weights $\mathbf{w}_{\text{init}}$

$\mathbf{c}_{init} = \mathbf{1}$ ▷ Initialise mask with ones
$\mathbf{w}_0 = \mathbf{c}_{\text{init}} \odot \mathbf{w}_{\text{init}}$ ▷ Apply mask to initial weights
for $k = 0, \ldots, M-1$ **do**
    $\mathcal{D}_k \sim \mathcal{D}^{\text{train}}$ ▷ Sample batch of data
    $\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_i, \mathcal{D}_k)$ ▷ Update network weights
end for

$g_j(\mathbf{w}, \mathcal{D}) = \partial \mathcal{L}(\mathbf{w}_M, \mathcal{D}) / \partial c_j$ ▷ Compute meta-gradient
$s_j = \frac{|g_j(\mathbf{w}, \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}, \mathcal{D})|}$ ▷ Compute saliency scores
Determine the k-th largest element in $\mathbf{s}$, s_k.
$\mathbf{c}_{\text{prune}} = \begin{cases} 1, & \text{if } c_j \geq s_k \\ 0, & \text{otherwise} \end{cases}$ ▷ Set pruning mask
$\hat{\mathbf{w}}_0 = \mathbf{c}_{\text{prune}} \odot \mathbf{w}_{\text{init}}$ ▷ Apply mask to initial weights $\mathbf{w}_{\text{init}}$

---

for $i = 1, \ldots, N$ **do** ▷ Train pruned model
    $\hat{\mathbf{w}}_{i+1} = \hat{\mathbf{w}}_i - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{w}}_i, \mathcal{D})$
end for

is:

$$\nabla_{\mathbf{c}} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M) = \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M) \left[\prod_{m=1}^{M} (\nabla_{\mathbf{w}_{m-1}} \mathbf{w}_m) \right] \mathbf{w}_{\text{init}}, \tag{3.16}$$

writing $\mathbf{w}_m$ in terms of $\mathbf{w}_{m-1}$ following SGD,

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M) \left[\prod_{m=1}^{M} \nabla_{\mathbf{w}_{m-1}} (\mathbf{w}_{m-1} - \alpha \nabla_{\mathbf{w}_{m-1}} \mathcal{L}(\mathbf{w}_{m-1}; \mathcal{D}_m)) \right] \mathbf{w}_{\text{init}}, \tag{3.17}$$

carrying through the partial derivative,

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M) \left[\prod_{m=1}^{M} I - \alpha \nabla^2_{\mathbf{w}_{m-1}} \mathcal{L}(\mathbf{w}_{m-1}; \mathcal{D}_m) \right] \mathbf{w}_{\text{init}}, \tag{3.18}$$

and finally dropping small terms for sufficiently small learning rates,

$$\approx \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M) \left[\prod_{m=1}^{M} I \right] \mathbf{w}_{\text{init}}, \tag{3.19}$$

$$= \nabla_{\mathbf{w}_M} \mathcal{L}(\mathbf{w}_M, \mathcal{D}_M)\ \mathbf{w}_{\text{init}}. \tag{3.20}$$

In the second-to-last step, we drop the higher-order terms, which gives us a first-order approximation of the meta-gradient[3].

With this approximation, we only need to save the initial weight vector $\mathbf{w}_{\text{init}}$ in memory and multiply it with the final gradient. This approximation can be crude when the Laplacian terms are large, but with a sufficiently small learning rate it becomes precise. The approximation allows us to take many more intermediate gradient-steps which can be beneficial for performance when the training dataset has many classes, as we will see in Section 3.4.2.

3.4 Experimental Results

3.4.1 3SP Experiments

In this section, we show that models pruned with 3SP are substantially faster than unpruned models and models pruned using baseline pruning-at-initialization methods without sacrificing much in accuracy. We show that a compute aware version (3SP + CA) can be even faster, but with a more significant reduction in performance. Finally, we demonstrate how this capability might be beneficial in a pratical setting where training speed is a bottleneck: we show that a 3SP model can achieve higher accuracy than a full model with a time-budget in active learning.

Baselines. We compare to previously published unstructured pruning-at-initialization methods SNIP (Lee et al., 2019d) and GraSP (Wang et al., 2020a), though we note that these methods do not provide a speed-up. We do not compare to methods that prune after or during training, because these incur a significant upfront computational cost, while this paper is focussed on constraining the *training* cost.

[3]Note that this approximation also works for optimizers other than vanilla SGD (e.g., Adam, Adamw, Adabound), except that the term which is dropped (r.h.s. of Equation Equation (3.18)) looks slightly different.

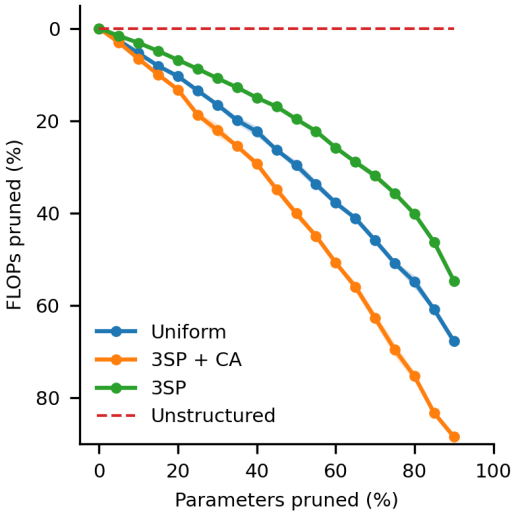


Figure 3.5: The relation between pruning parameters and the effect on compute on CIFAR-10 with VGG-19. Uniform is a naive baseline which prunes all channels with uniform probability, and therefore ignores both model loss and compute cost. 3SP removes channels least important to the loss, which happen to also use less compute on average. 3SP + CA, without compute smoothing, actively prunes FLOPs while taking model performance into account (see Figure 3.6).

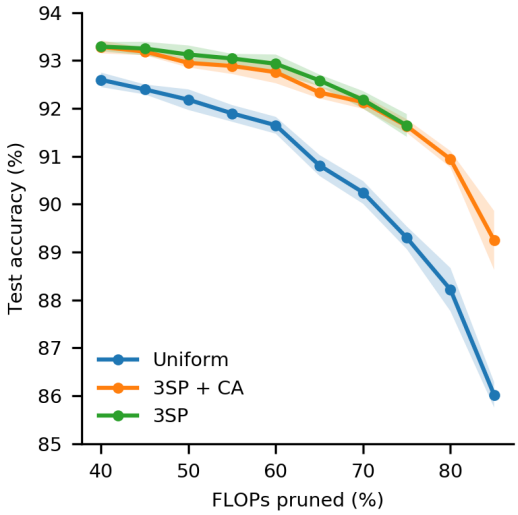


Figure 3.6: The performance of 3SP and 3SP + CA (compute-aware) against a baseline of uniformly removing channels from the model on CIFAR-10 with VGG-19. At every amount of compute pruned, 3SP outperforms uniform, with the difference increasing as more FLOPs are pruned. After removing 75% of FLOPs, 3SP becomes unable to prune more without pruning entire layers. Our compute-aware extension, however, is able to remove even more compute without failing.

3SP and its compute-aware extension are the first methods to structurally prune a model before training, showing that this approach is feasible and a ground for further research.

We also compare to a naive structured pruning method which uniformly prunes over all mask entries and layers. This "Uniform" baseline samples entries for binary mask tensors $\mathbf{c}$ or $\tilde{\mathbf{c}}$ from Bern(p) in order to prune a p'th of the weights. In expectation, the model width is pruned by a uniform ratio in each layer. This is similar to using a narrower model. Liu et al. (2019) showed that these sorts of untargeted pruning methods are effective baselines that are able to obtain strong performance.

Architecture and random seeds. Our experiments are executed using VGG-19; we describe the exact architecture in Appendix A.1. We run all our experiments 5

Table 3.1: Accuracy of 3SP (grey), SNIP and GraSP for VGG19 on CIFAR-10 and CIFAR-100. We evaluate on the basis of pruned parameters because prior methods are unstructured and do not reduce compute cost. However, the speed-based evaluation of Figure 3.6 is a much better view of our method's performance. Even on the basis of parameter sparsity, 3SP performs comparably with unstructured methods on CIFAR-10 though a gap forms on the more complex CIFAR-100. Rescaling/reinitializing has little effect on the smaller dataset, but appears to matter for CIFAR-100. The original (unpruned) model obtains 93.6% accuracy on CIFAR-10 and 72.5% on CIFAR-100.

Method		CIFAR-10			CIFAR-100		
		Parameters Pruned (Acc. ± s.e.)			Parameters Pruned (Acc. ± s.e.)		
		80%	90%	95%	80%	90%	95%
Unstructured	Uniform	92.6 ± 0.04 %	91.4 ± 0.04 %	89.8 ± 0.04 %	70.3 ± 0.16 %	68.1 ± 0.11 %	64.7 ± 0.27 %
	SNIP	93.6 ± 0.12 %	93.6 ± 0.05 %	93.4 ± 0.04 %	72.8 ± 0.10 %	72.4 ± 0.08 %	70.7 ± 0.11 %
	GraSP	93.2 ± 0.09 %	93.0 ± 0.03 %	92.8 ± 0.08 %	71.2 ± 0.08 %	70.6 ± 0.15 %	69.5 ± 0.07 %
Structured	Uniform	92.0 ± 0.08 %	90.4 ± 0.12 %	89.0 ± 0.15 %	67.5 ± 0.16 %	63.8 ± 0.13 %	60.1 ± 0.29 %
	3SP	93.4 ± 0.03 %	93.1 ± 0.04 %	92.5 ± 0.12 %	69.9 ± 0.14 %	68.3 ± 0.12 %	63.2 ± 0.52 %
	3SP + re-init	93.4 ± 0.04 %	93.0 ± 0.02 %	92.6 ± 0.09 %	70.3 ± 0.16 %	69.0 ± 0.08 %	64.2 ± 0.35 %
	3SP + re-scale	93.3 ± 0.03 %	93.0 ± 0.06 %	92.5 ± 0.06 %	70.5 ± 0.13 %	69.2 ± 0.11 %	63.5 ± 0.63 %

times using different random seeds, reporting the mean and standard error of each experiment in tables (standard deviation in figures, as s.e. was not visible).

Measuring compute. We report model compute cost in Floating Point operations (FLOPs). The exact number of FLOPs used to perform a calculation depends on the hardware, so we make the common assumption that roughly two FLOPs are required for each multiply-accumulate operation.

Compute-cost vs. Accuracy Trade-off. In Figure 3.5 we show that our approach is able to reduce the FLOPs required for a forward pass in the model. 3SP tries to preserve model performance, so it removes compute less aggressively than the Uniform baseline. However, 3SP + CA (compute-aware) with $\lambda = 0$ is able to very aggressively remove compute. In Figure 3.6 we show the trade-off between the compute cost of the model and accuracy for 3SP, 3SP + CA and Uniform pruning on CIFAR10. We show that there is a significant gap between 3SP and uniform, indicating that we are able to successfully prune our model. When trying to prune very large amounts of compute, having ignored compute costs during pruning, 3SP is forced to remove entire layers. Our compute aware extension, however, is able to continue pruning to very high levels of compute sparsity.

Parameter vs. Accuracy Trade-off. In Table 3.1 we consider accuracy when pruning different proportions of *parameters* using VGG-19 (Simonyan et al., 2015)

Table 3.2: Effect of the computational cost reduction on training/inference time. Reducing FLOPs directly reduces training time and prediction time, and indirectly reduces the model size. These results are obtained using the VGG-19 model, CIFAR10 dataset, and a GTX 1080 Ti GPU. The prediction time is measured using a batch with a single element on a i7 8700K CPU. The pruning time is equal to one step in the original model.

	Original	50% FLOPs Reduction	80% FLOPs Reduction
Epoch Training Time (GPU)	15 s	8 s (~2x)	4 s (~4x)
Prediction Time (CPU)	12.8 ms	4.2 ms (~3x)	2.6 ms (~5x)
Model Size	87 MB	12 MB (~7x)	3.3 MB (~26x)
Pruning Time	–	41 ms	41 ms
Accuracy (CIFAR-10)	93.6% ± 0.04	93.1% ± 0.09	90.9 ± 0.09%

on CIFAR-10 and CIFAR-100. As we have discussed, removing parameters is not as important as reducing compute cost, which is why the best evaluation of our method is that provided in Figure 3.6. But the only prior pruning-at-initialization methods are unstructured and lead to no compute improvement at all. Therefore, in order to compare to prior methods, we show in Table 3.1 that even when looking at parameter-sparsity, 3SP peforms nearly as well as unstructured methods for CIFAR-10. At 80% parameter sparsity 3SP has similar accuracy, and even at 95% accuracy 3SP is less than a percentage point less accurate than the unstructured SNIP and GraSP methods. For CIFAR-100, the structured constraint has a bigger effect on performance, though 3SP still performs significantly better than Uniform pruning. This can be explained by the fact that CIFAR-100 is a more difficult dataset, and VGG19 is therefore less overparameterized. Re-initializing the weights has a very small effect on the accuracy on CIFAR-10 and helps with CIFAR-100. This suggests that these pruning before training methods predominantly identify an important model architecture rather than individual weights.

Wall Clock Time and Model Size. In Table 3.2, we show the tangible benefits of using structured pruning. The 50% reduction model is without compute aware, while the 80% model is with compute aware and smoothing set to 0.05. On a GPU, reducing the FLOPs by 80% or 50% leads to a 4x or 2x speed-up in training time per epoch. On a CPU the benefit is even greater. Because structured pruning results in a much smaller model, and induces a smaller activation map, the forward pass can be kept almost entirely in CPU cache, leading to a 5x or 3x speedup with the same levels of pruning instead. The smaller memory footprint of our method would

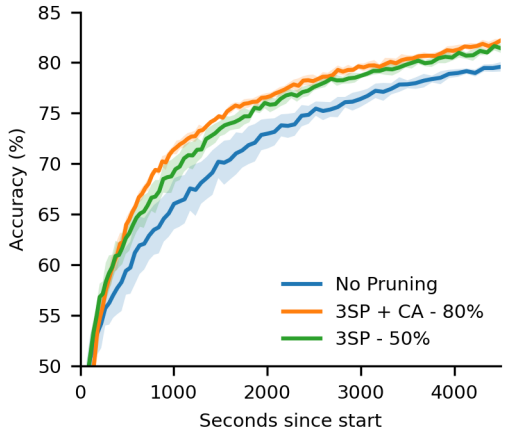


Figure 3.7: Active Learning CIFAR-10. 3SP model at 50% and 3SP-CA at 80% FLOPs reduction have better accuracy than a full model within a time-budget because they train, and therefore acquire data, more quickly.

allow pruned models to be used in settings that unstructured pruning might not help with. Adopting our method can therefore help researchers and organizations reduce run-time and power consumption of training, evaluating, and deploying their models, leading to a reduced carbon footprint. Full details are provided in Appendix A.2.

Active Learning with 3SP Pruning

In some settings, we have a large pool of unlabelled data for which we can request labels, but the labelling process is costly (for example, requiring highly-trained experts). Active learning (Settles, 2010) attempts to pick the most informative datapoints out of the pool to reduce the labelling cost. Unfortunately, most active learning approaches require retraining the model on the labelled set before acquiring new points. This can mean that the model must be trained tens or hundreds of times during the acquisition process. We show that a 3SP model can reach a higher accuracy within a time-budget than an unpruned model (Figure 3.7). Even though pruning hurts model performance, the fact that we prune before training in a structured way lets us train much more quickly. This lets the 3SP model acquire more data within the same time-budget, which allows higher accuracy. Full experimental details are provided in Appendix A.3.

Alternative Pruning Criterion - GraSP

Wang et al. (2020a) propose to keep network parameters that contribute the most to loss reduction during optimization. Instead of using the change in loss as a pruning

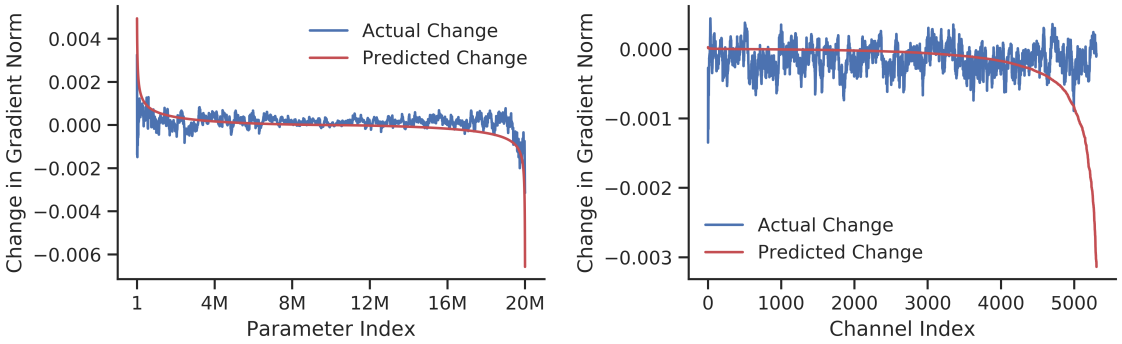
score, like SNIP, they use the predicted change in the magnitude of the gradient with respect to the weights, approximated with a first-order Taylor expansion. In order to avoid finding a mask that satisfies their objective trivially by creating a very large loss, they strongly smooth outputs so that removing weights cannot greatly change the loss. Although Wang et al. (2020a) motivate their method by the fact that their score measures the interaction between weights, we observe that it is still a first-order method and does not assess the interaction between the decision to *prune* multiple weights.

The naive form of GraSP cannot be directly applied to structured models because GraSP computes gradients with respect to weights directly, not mask variables. But we consider a structured method inspired by GraSP which instead computes gradients with respect to mask tensor entries. We found in our experiments that this objective performed substantially worse than one based on SNIP. On CIFAR10, structured GraSP achieves accuracies of 91.96%, 91.6%, and 90.92% at 80-90-95% prune ratios respectively, worse than Uniform pruning (compare to Table 3.1).

We believe that this is because estimates of the GraSP-style objective are too noisy in the structured setting, and so assumption 2 considered in §3.2 is violated. Similar to the experiment in Figure 3.2, we compare the predictions implied by the gradient to the actual effect on the objective. Figure 3.8 shows that in the unstructured case, there is a correlation between the approximated change in the objective and the actual change. However, the structured prediction is uncorrelated with the actual change.

3.4.2 ProsPr Experiments

We empirically evaluate the performance of our method, ProsPr, compared to various vision classification baselines across different architectures and datasets. In supplementary sections we show effectiveness of our method on image segmentation tasks (Appendix A.7) and when using self-supervised initialization (Appendix A.8). We provide details of our hyper-parameters, experiment setup, and implementation details in Appendix A.4.



(a) Unstructured **(b)** Structured

Figure 3.8: Evaluating approximation accuracy of GraSP objective for structured and unstructured pruning of VGG-19. We compare the predicted change in the gradient-norm given by the Taylor expansion (scores) with the actual change in the gradient-norm (normalized). The unstructured predictions have signal, but in the structured setting they are too noisy.

Results on CIFAR and Tiny-Imagenet

In recent work, Frankle et al. (2021) extensively study and evaluate different pruning-at-initialization methods under various effects such as weight re-initialization, weight shuffling, and score inversion. They report the best achievable results by these methods and highlight the gap between their performance and two pruning-at-convergence methods, weight rewinding and magnitude pruning (Renda et al., 2020; Frankle et al., 2020).

In Figure 3.9 we evaluate ProsPr on this benchmark using ResNet-20 and VGG-16 on CIFAR-10, and ResNet-18 on Tiny-ImageNet. It can be seen that ProsPr reduces the performance gap, especially at higher sparsity levels, and in some cases exceeds the accuracy of pruning-after-convergence methods. Full results are also summarized in Appendix A.5.

This is a remarkable step forward in closing the gap to methods that prune after training. Previous works that prune at the start have not been able to outperform methods that prune after training on any settings, including smaller datasets such as CIFAR-10 or Tiny-ImageNet. It is also important to note that other baselines that have comparable accuracies are all iterative methods. ProsPr is the only method that can do this in a single shot manner by taking only 3 additional steps in the inner-loop before computing the meta-gradients. In total, we only use 4 batches of

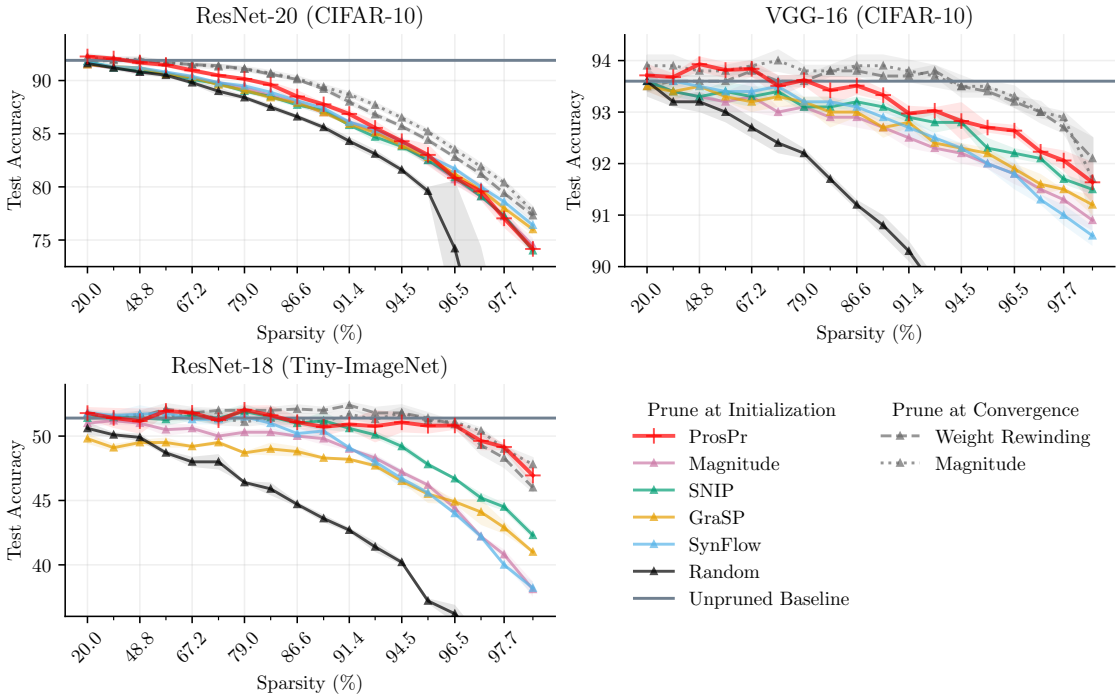


Figure 3.9: Accuracy of ProsPr against other prune-at-init and prune-after-convergence methods as benchmarked by Frankle et al. (2021). The shaded areas denote the standard deviation of the runs.

data. We also do not do any averaging of scores by repeating the method multiple times.

The performance in these small datasets comes from the fact that ProsPr computes higher-order gradients. While there are other iterative methods that can work without any data, their effect is mostly a more graceful degradation at extreme pruning ratios as opposed to best accuracy at more practical sparsity levels. One example is SynFlow which is similar to FORCE but uses an all-one input tensor instead of samples from the training set (Tanaka et al., 2020).

Results on ImageNet dataset

To evaluate the performance of ProsPr on more difficult tasks we run experiments on the larger ImageNet dataset. Extending gradient-based pruning methods to this dataset poses several challenges.

Number of classes. In synaptic-saliency methods, the mini batches must have

Table 3.3: Test accuracies of VGG-19 and ResNet-50 on ImageNet. First-Order ProsPR exceeds the results reported by Jorge et al. (2021) in all configurations but one, where GraSP works best.

	VGG-19				ResNet-50			
Sparsity	90%		95%		90%		95%	
Accuracy	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Unpruned Baseline	73.1	91.3	—	—	75.6	92.8	—	—
ProsPr (ours)	**70.75**	**89.9**	66.1	87.2	**66.86**	**87.88**	**59.62**	**82.82**
FORCE	70.2	89.5	65.8	86.8	64.9	86.5	59.0	82.3
Iter-SNIP	69.8	89.5	65.9	86.9	63.7	85.5	54.7	78.9
GRASP-MB	69.5	89.2	**67.6**	**87.8**	65.4	86.7	46.2	66.0
SNIP-MB	68.5	88.8	63.8	86.0	61.5	83.9	44.3	69.6
SNIP-MB-train	0.00	0.00	0.00	0.00	66.5	87.3	59.4	82.5
Random	64.2	86.0	56.6	81.0	64.6	86.0	57.2	80.8

enough examples from all classes in the dataset. Wang et al. (2020a) recommend using class-balanced mini-batches sized ten times the number of classes. In datasets with few classes this is not an issue and even a single batch includes multiple examples per class. This is one reason why methods like SNIP work with a single batch, and why we kept the number of steps in ProsPr's inner loop fixed to only 3. ImageNet however has 1,000 classes, and using a single or a handful of small batches is inadequate. Previous methods such as FORCE, GraSP, or SynFlow avoid this problem by repeating the algorithm with new data batches and averaging the saliency scores. In ProsPr we instead increase the number of updates before computing the meta-gradients, ensuring they flow through enough data. Computing meta-gradients through many steps however poses new challenges.

Gradient degradation. We start to see gradient stability issues when computing gradients over deep loops. Gradient degradation problems, i.e., vanishing and exploding gradients, have also been observed in other fields that use meta-gradients such as Meta-Learning. Many solutions have been proposed to stabilize gradients when the length of loop increases beyond 4 or 5 steps, although this remains an open area of research (Antoniou et al., 2019).

Computation Complexity. For ImageNet we must make the inner loop hundreds of steps deep to achieve balanced data representation. In addition to stability issues, backpropagating through hundreds of steps is very compute intensive.

Therefore for our experiments on ImageNet we use the first-order approximation of ProsPr (Sec 3.3.2). We evaluate ProsPr using ResNet-50 and VGG-19 architectures and compare against state-of-the-art methods FORCE and Iter-SNIP introduced by Jorge et al. (2021). We include multi-batch versions of SNIP and GraSP (SNIP-MB and GraSP-MB) to provide a fair comparison to iterative methods, which partially prune several times during training, in terms of the amount of data presented to the method. We use 1024 steps with a batch size of 256 (i.e. 262,144 samples) for ResNet-50. For VGG-19, a much larger model, and which requires more GPU memory we do 256 steps with batch size of 128. This is still far fewer samples than other methods. Force, for example, gradually prunes in 60 steps, where each step involves computing and averaging scores over 40 batches of size 256, i.e. performing backpropagation 2400 times and showing 614,400 samples to the algorithm.

Table 3.3 shows our results compared to the baselines reported by Jorge et al. (2021). First-order ProsPr exceeds previous results in all configurations except one, where it is outperformed by GraSP. Note the surprisingly good performance of random pruning of ResNets, compared to other methods. This was also observed by Jorge et al. (2021). This could be explained by the fact that VGG-19 is a much larger architecture with 143.6 million parameters, compared to 15.5 million in ResNet-50s. More specifically the final three dense layers of VGG-19 constitute 86% of its total prunable parameters. The convolution layers of VGG constitute only 14% of the prunable weights. Pruning methods are therefore able to keep more of the convolution weights and instead prune extensively from the over-parametrized dense layers. ResNet architectures on the other hand have a single dense classifier at the end.

Structured Pruning

We also evaluate ProsPr in the structured pruning setup where instead of pruning individual weights, entire convolutional channels (or columns of linear layers) are removed. This is a significantly more restricted setup but in addition to memory savings also reduces the computational cost of training and inference.

Adopting ProsPr for structured pruning is as simple as changing the shape of the pruning mask $\mathbf{c}$ in Eq 3.6 to have one entry per channel (or column of the weight matrix). We evaluate our method against 3SP, a method that extends SNIP to

structured pruning introduced in §3.2. Our results are summarized in Table 3.4 which show accuracy improvements in all scenarios. In Appendix A.6 we also evaluate wall-clock improvements in training time as a result of structured pruning at initialization.

Table 3.4: Test accuracies for structured pruning using VGG-19 on CIFAR-10 and CIFAR-100. ProsPr achieves better accuracy in all configurations.

Sparsity	Method	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)
—	Unpruned Baseline	93.6	72.5
80%	ProsPr (ours)	**93.61 ± 0.01**	**72.29 ± 0.11**
	3SP	93.4 ± 0.03	69.9 ± 0.14
	3SP + reinit	93.4 ± 0.04	70.3 ± 0.16
	3SP + rescale	93.3 ± 0.03	70.5 ± 0.13
	Random	92.0 ± 0.08	67.5 ± 0.16
90%	ProsPr (ours)	**93.64 ± 0.24**	**71.12 ± 0.26**
	3SP	93.1 ± 0.04	68.3 ± 0.12
	3SP + reinit	93.0 ± 0.02	69.0 ± 0.08
	3SP + rescale	93.0 ± 0.06	69.2 ± 0.11
	Random	90.4 ± 0.12	63.8 ± 0.13
95%	ProsPr (ours)	**93.32 ± 0.15**	**68.03 ± 0.38**
	3SP	92.5 ± 0.12	63.2 ± 0.52
	3SP + reinit	92.6 ± 0.09	64.2 ± 0.35
	3SP + rescale	92.5 ± 0.06	63.5 ± 0.63
	Random	89.0 ± 0.15	60.1 ± 0.29

Number of Meta Steps

Finally, we evaluate ProsPr when using a varying number of meta steps, which gives insight into whether using meta-gradients is beneficial. We repeated experiments from Section 3.4.2 but this time we vary the depth of training steps between 0 and 3. The results in Table 3.5 show that the final accuracy consistently increases as we increase the depth of the training, showing the effectiveness of meta-gradients. We used the same data batch in all M training steps to isolate the effect of M, while in other experiments we use a new batch in every step.

In theory increasing the number of training steps should always help and in the limit of large M, the match to the ultimate objective (estimating the loss after many

Table 3.5: Evaluating the effect of meta steps (M) on structured pruning performance of VGG-19

(a) 90% Sparsity

M	CIFAR-10 Acc	CIFAR-100 Acc
0	93.1% ± 0.04	68.3% ± 0.12
1	93.3% ± 0.12	68.8% ± 0.18
2	93.5% ± 0.21	69.8% ± 0.20
3	93.6% ± 0.19	71.0% ± 0.21

(b) 95% Sparsity

M	CIFAR-10 Acc	CIFAR-100 Acc
0	92.5% ± 0.12	63.20% ± 0.52
1	92.9% ± 0.31	64.87% ± 0.35
2	93.25% ± 0.24	67.12% ± 0.42
3	93.29% ± 0.29	67.98% ± 0.31

epochs of training) would be exact. However, in practice increasing the number of steps beyond 3 poses a lot of gradient stability issues (and is computationally expensive). These issues have been also identified in the meta-learning literature and various solutions have been proposed to fix it (Antoniou et al., 2019).

3.5 Related Work

Pruning at initialization Several works build on top of the pruning-at-initialization approach from Lee et al. (2019d): Wang et al. (2020a) propose an alternative criterion to minimizing changes in the loss and instead argue for preserving the gradient flow. Their method, GraSP, keeps weights that contribute most to the *norm* of the gradients. Jorge et al. (2021) evaluate the SNIP objective in a loop in which pruned parameters still receive gradients and therefore have a chance to get *un*-pruned. The gradual pruning helps avoid the layer-collapse issue, and their method, known as FORCE, achieves more graceful performance-degradation at extreme sparsity levels. Tanaka et al. (2020) provide theoretical justification for why iteratively pruning helps with the layer-collapse issue and propose a *data-free* version of the method where an all-one input tensor is used instead of real training data.

Hayou et al. (2021) show that a pruned model is trainable only when the initialization is on the Edge-of-Chaos (Schoenholz et al., 2016) and propose a rescaling method for architectures that do not satisfy the condition. Lee et al. (2019c) analyze the signal-propagation properties of initialization and propose using orthogonal weights to enforce dynamical isometry in order to obtain reliable connection sensitivities.

Neural Architecture Search (NAS) is also related to our work insofar as it optimizes a network architecture before training. However, our work (3SP) requires only one forwards-backwards pass on a single batch to prune, rather than requiring extensive retraining. In general, NAS methods have been accused of consuming enormous amounts of energy (Strubell et al., 2019), counter to the goal of this paper.

Pruning during training As discussed in Section 3.1, in existing methods the training step has been absent from the saliency computation step. As a workaround, many methods make their approaches *training-aware* by applying pruning gradually and interleaving it with training: A simple but effective pruning-during-training scheme was proposed by Zhu et al. (2018), and more recently Gale et al. (2019) showed its effectiveness in a broader range of tasks. Lym et al. (2019) continuously apply structured pruning via group-lasso regularization while at the same time increasing batch sizes. In a related work to ours, You et al. (2020) propose finding stable, trainable, pruned architectures after 20+ epochs of training and pruning and monitoring a distance metric. Frankle et al. (2019) show that weight rewinding achieves better results when done in multiple prune-retrain steps.

Wang et al. (2020b) propose making the starting model even larger by increasing the convolutional channels uniformly, and formulate pruning-at-initialization as a Neural Architecture Search (NAS) within this larger search space. They propose a channel-wise mask similar to our method and directly optimize it using a sparsity penalty similar to Liu et al. (2017). Moreover, following the recipe proposed by Liu et al. (2019), they keep the training-time budget constant by increasing the number of epochs.

Compute-aware pruning Within the pruning-after-training literature, there are a number of methods that explicitly consider computational cost and others that apply structured pruning. Gordon et al. (2018) explicitly regularize FLOPS while training with a sparsity penalty, while Veniat et al. (2018) and Theis et al. (2018) adopt a similar method motivated as approximate constrained optimization. He et al. (2019) frame model crafting as a reinforcement learning problem where the reward is based on compute usage. Many methods for pruning after training impose a structured pruning mask, which results in compute savings (Li et al., 2016; He et al., 2017; Liu et al., 2017; Luo et al., 2017). These methods all train using

the full-sized network and often require significant finetuning, sometimes with as many epochs as used during training. For example Wang et al. (2019) obtains very strong results but needs 160 epochs of training a full-sized network and another 160 epochs of fine-tuning.

3.6 Summary

We started this chapter by introducing an extension of prune-at-initialization methods to structured pruning. We show for the first time that structured pruning at initialization is feasible and can be an exciting way to reduce wasteful training costs and architecture selection time. We also introduced and evaluated a compute-aware variant that leads to even more speed-ups at the expense of accuracy.

However, this method did not quite live up to its promise, especially compared to prune-after-training methods. We argued that this is, in part, because it does not account for the fact that the pruned network is going to be *trained* after it is pruned. We showed how using meta-gradients as a pruning signal takes us closer to capturing the effect of a pruning mask on the training procedure. As a result, our meta-gradient method is competitive not just with methods that prune before training but also with methods that prune iteratively during training and those that prune post-training.

In the next chapter we shift our attention to the other popular compression approach for compressing neural networks, i.e. quantization, and show how meta-gradients can be used to *drive* training towards learning models with inherent robustness against post-training quantization.

Chapter 4

Quantization Robustness Using Meta-Gradients

In the previous chapter we showed how meta-gradients can be used as *signal* to identify trainable subnetworks for pruning neural networks at initialization. In this chapter we change our focus to the problem of quantization and show how meta-gradients can be used in a regularization scheme to *learn* models with inherent robustness against post-training quantization.

4.1 Background

Quantized neural networks allow for more speed and energy efficiency compared to floating-point models by using fixed-point arithmetic but naive quantization of pre-trained models often results in severe accuracy degradation, especially when targeting bit-widths below eight (Krishnamoorthi, 2018a). Performant quantized models can be obtained via quantization-aware training or fine-tuning, i.e., learning full-precision *shadow* weights for each weight matrix with backpropagation using the straight-through estimator (STE) (Bengio et al., 2013b), or using other approximations (Louizos et al., 2018). Alternatively, there have been successful attempts to recover the lost model accuracy without requiring a training pipeline (Banner et al., 2018; Meller et al., 2019; Choukroun et al., 2019; Zhao et al., 2019) or representative data (Nagel et al., 2019b).

But these methods are not without drawbacks. The shadow weights learned through quantization-aware fine-tuning often do not show robustness when quantized to bit-widths other than the one they were trained for (see Table 4.1). In practice, the training procedure has to be repeated for each quantization target. Furthermore, post-training recovery methods require intimate knowledge of the relevant architectures. While this may not be an issue for the developers training the model in the first place, it is a difficult step for middle parties that are interested in picking up models and deploying them to users down the line, e.g., as part of a mobile app. In such cases, one might be interested in automatically constraining the computational complexity of the network such that it conforms to specific battery consumption requirements, e.g. employ a 4-bit variant of the model when the battery is less than 20% but the full precision one when the battery is over 80%. Therefore, a model that can be quantized to a specific bit-width "on the fly" without worrying about quantization aware fine-tuning is highly desirable.

In this chapter, we explore a novel route, substantially different from the methods described above. We start by investigating the theoretical properties of noise introduced by quantization and analyze it as a ℓ_∞-bounded perturbation. Using this analysis, we derive a straightforward regularization scheme based on meta-gradients to *control* the maximum first-order induced loss and learn networks that are inherently more robust against post-training quantization. We show that applying this regularization at the final stages of training, or as a fine-tuning step after training, improves post-training quantization across different bit-widths at the same time for commonly used neural network architectures.

By training quantization-ready networks, our approach enables storing a single set of weights that can be quantized on-demand to different bit-widths as energy and memory requirements of the application change. Unlike quantization-aware training using the straight-through estimator that only targets a specific bit-width and requires access to training data and pipeline, our regularization-based method paves the way for "on the fly" post-training quantization to various bit-widths. We show that by modeling quantization as a ℓ_∞-bounded perturbation, the first-order term in the loss expansion can be regularized using the ℓ_1-norm of gradients.

4.2 Modelling and Controlling Quantization

4.2.1 Quantization Noise

In this section, we propose an appropriate model for quantization noise. Then, we show how we can effectively control the first-order, i.e., the linear part of the output perturbation caused by quantization. When the linear approximation is adequate, our approach provides robustness towards various quantization bit-widths simultaneously.

We use the following notation throughout the paper. The ℓ_p-norm of a vector $\boldsymbol{x}$ in $\mathbb{R}^n$ is denoted by $\|\boldsymbol{x}\|_p$ and defined as $\|\boldsymbol{x}\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$ for $p \in [1, \infty)$. At its limit we obtain the ℓ_∞-norm defined by $\|\boldsymbol{x}\|_\infty := \max_i |x_i|$. The inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

The error introduced by rounding in the quantization operation can be modeled as a generic additive perturbation. Regardless of which bit-width is used, the quantization perturbation that is added to each value has bounded support, which is determined by the width of the quantization bins. In other words, the quantization noise vector of weights and activations in neural networks has entries that are bounded. Denote the quantization noise vector by $\boldsymbol{\Delta}$. If δ is the width of the quantization bin, the vector $\boldsymbol{\Delta}$ satisfies $\|\boldsymbol{\Delta}\|_\infty \leq \delta/2$. Therefore we model the quantization noise as a perturbation bounded in the ℓ_∞-norm. A model robust to ℓ_∞-type perturbations would also be robust to quantization noise.

To characterize the effect of perturbations on the output of a function, we look at its tractable approximations. To start, consider the first-order Taylor-expansion of a real valued-function $f(\boldsymbol{w} + \boldsymbol{\Delta})$ around $\boldsymbol{w}$:

$$f(\boldsymbol{w} + \boldsymbol{\Delta}) = f(\boldsymbol{w}) + \langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w}) \rangle + R_2, \tag{4.1}$$

where R_2 refers to the higher-order residual error of the expansion. We set R_2 aside for the moment and consider the output perturbation appearing in the first-order term $\langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w}) \rangle$. The maximum of the first-order term among all ℓ_∞-bounded perturbations $\boldsymbol{\Delta}$ is given by:

$$\max_{\|\boldsymbol{\Delta}\|_\infty \leq \delta} \langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w}) \rangle = \delta \left\| \nabla f(\boldsymbol{w}) \right\|_1. \tag{4.2}$$

To prove this, consider the inner product of $\boldsymbol{\Delta}$ and an arbitrary vector $\boldsymbol{x}$ given by $\sum_{i=1}^{n} n_i x_i$. Since $|n_i|$ is assumed to be bounded by δ, each $n_i x_i$ is bounded by $\delta|x_i|$, which yields the result. The maximum in Equation 4.2 is obtained indeed by choosing $\boldsymbol{\Delta} = \delta \operatorname{sign}(\nabla f(\boldsymbol{w}))$.

Equation 4.2 comes with a clear hint. We can guarantee that the first-order perturbation term is small if the ℓ_1-norm of the gradient is small. In this way, the first-order perturbation can be controlled efficiently for various values of δ, i.e. for various quantization bit-widths. In other words, an effective way for controlling the quantization robustness, up to first-order perturbations, is to control the ℓ_1-norm of the gradient. As we will shortly argue, this approach yields models with the best robustness.

This conclusion is based on worst-case analysis since it minimizes the upper bound of the first-order term, which is realized by the worst-case perturbation. Its advantage, however, lies in simultaneous control of the output perturbation for all δs and all input perturbations. In the context of quantization, this implies that the first-order robustness obtained in this way would hold regardless of the adopted quantization bit-width or quantization scheme.

The robustness obtained in this way would persist even if the perturbation is bounded in other ℓ_p-norms. The robustness to ℓ_∞-norm perturbations is, therefore, the most stringent one among other ℓ_p-norms, because a model should be robust to a broader set of perturbations. Controlling the ℓ_1-norm of the gradient guarantees robustness to ℓ_∞-perturbations and thereby to all other ℓ_p-bounded perturbations.

4.2.2 Regularization With Meta-Gradients

In what follows, we propose regularizing the ℓ_1-norm of the gradient to promote robustness to bounded norm perturbations and in particular bounded ℓ_∞-norm perturbations. These perturbations arise from quantization of weights and activations of neural networks.

We focused on weight quantization in our discussions so far, but we can equally apply the same arguments for activation quantization. Although the activations are not directly learnable, their quantization acts as an additive ℓ_∞-bounded perturbation on their outputs. The gradient of these outputs is available. It therefore suffices to

accumulate all gradients along the way to form a large vector for regularization.

Suppose that the loss function for a deep neural network is given by $L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x})$ where $\mathbb{W}$ denotes the set of all weights, $\mathbb{Y}$ denotes the set of outputs of each activation and $\boldsymbol{x}$ the input. We control the ℓ_1-norm of the gradient by adding the regularization term

$$\sum_{\boldsymbol{W}_l \in \mathbb{W}} \|\nabla_{\boldsymbol{W}_l} L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x})\|_1 + \sum_{\boldsymbol{y}_l \in \mathbb{Y}} \|\nabla_{\boldsymbol{y}_l} L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x})\|_1$$

to the loss, yielding an optimization target

$$L(\mathbb{W}; \boldsymbol{x}) = L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x}) + \lambda_w \sum_{\boldsymbol{W}_l, \in \mathbb{W}} \|\nabla_{\boldsymbol{W}_l} L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x})\|_1 + \lambda_y \sum_{\boldsymbol{y}_l \in \mathbb{Y}} \|\nabla_{\boldsymbol{y}_l} L_{CE}(\mathbb{W}, \mathbb{Y}; \boldsymbol{x})\|_1 , \tag{4.3}$$

where λ_w and λ_y are weighing hyper-parameters.

4.2.3 Alternative Regularizations

The equivalence of norms in finite-dimensional normed spaces implies that all norms are within a constant factor of one another. Therefore, one might suggest regularizing any norm to control other norms. Indeed some works attempted to promote robustness to quantization noise by controlling the ℓ_2-norm of the gradient (Hoffman et al., 2019). However, an argument related to the curse of dimensionality can show why this approach will not work. The equivalence of norms for ℓ_1 and ℓ_2 in n-dimensional space is stated by the inequality:

$$\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\, \|\boldsymbol{x}\|_2 .$$

Although the ℓ_2-norm bounds the ℓ_1-norm from above, it is vacuous if it does not scale with $1/\sqrt{n}$. Imposing such a scaling is demanding when n, which is the number of trainable parameters, is large. Figure 4.4 shows that there is a large discrepancy between these norms in a conventionally trained network, and therefore small ℓ_2-norm does not adequately control the ℓ_1-norm. A very similar argument can be provided from a theoretical perspective (see the supplementary materials).

To guarantee robustness, the ℓ_2-norm of the gradient, therefore, should be pushed as small as $\Theta(1/\sqrt{n})$. We experimentally show in Section 4.3 that this is a difficult task.

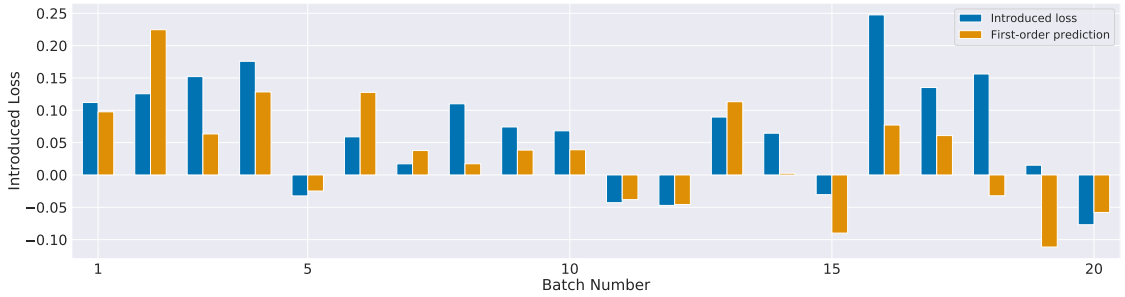


Figure 4.1: Predicting induced loss using first-order terms. We added ℓ_∞-bounded noise with δ corresponding to 4-bit quantization to all weights of ResNet-18 and compared the induced loss on the CIFAR-10 test-set with the predictions using gradients. While not perfect, the first-order term is not insignificant.

We therefore directly control the ℓ_1-norm in this paper. Note that small ℓ_1-norm is guaranteed to control the first order-perturbation for all types of quantization noise with bounded support. This includes symmetric and asymmetric quantization schemes.

Another concern is related to the consistency of the first-order analysis. We neglected the residual term R_2 in the expansion. Figure 4.1 compares the induced loss after perturbation with its first-order approximation. The approximation shows a strong correlation with the induced loss. We will see in the experiments that the quantization robustness can be boosted by merely controlling the first-order term. Nonetheless, a higher-order perturbation analysis can probably provide better approximations. Consider the second-order perturbation analysis:

$$f(\boldsymbol{w}+\boldsymbol{\Delta}) = f(\boldsymbol{w}) + \langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w}) \rangle + \frac{1}{2}\boldsymbol{\Delta}^T \nabla^2 f(\boldsymbol{w}) \boldsymbol{\Delta} + R_3.$$

Computing the worst-case second-order term for ℓ_∞-bounded perturbations is hard. Even for convex functions where $\nabla^2 f(\boldsymbol{w})$ is positive semi-definite, the problem of computing worst-case second-order perturbation is related to the mixed matrix-norm computation, which is known to be NP-hard. There is no polynomial-time algorithm that approximates this norm to some fixed relative precision (Hendrickx et al., 2010). For more discussions, see the supplementary materials. It is unclear how this norm should be controlled via regularization.

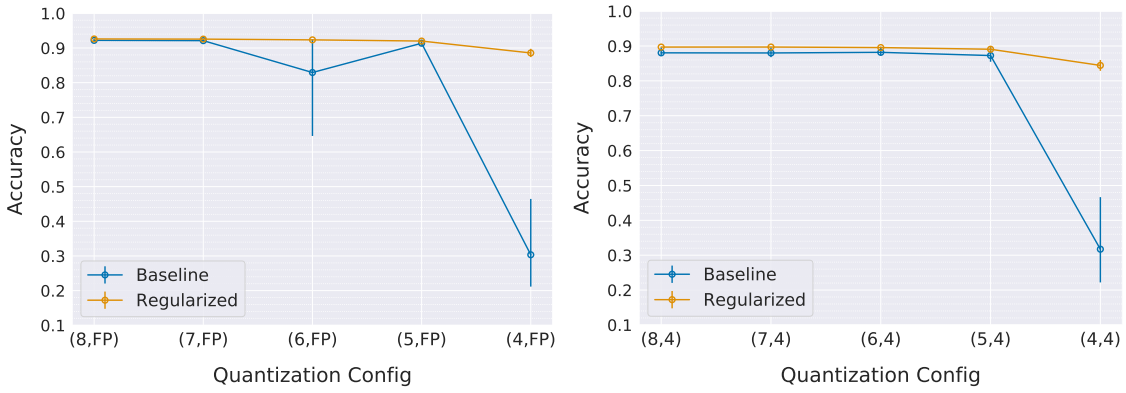


Figure 4.2: Accuracy of regularized VGG-like after post-training quantization. We trained 5 models with different initializations and show the mean accuracy for each quantization configuration. The error bars indicate min/max observed accuracies. (a) Weight-only quantization (b) Activation quantization fixed to 4-bits

4.3 Experimental Results

In this section we experimentally validate the effectiveness of our regularization method on improving post-training quantization. We use the well-known classification tasks of CIFAR-10 with ResNet-18 (He et al., 2016b) and VGG-like (Simonyan et al., 2014) and of ImageNet with ResNet-18. We compare our results for various bit-widths against (1) unregularized baseline networks (2) Lipschitz regularization methods (Lin et al., 2019; Gulrajani et al., 2017) and (3) quantization-aware fine-tuned models. Note that Gulrajani et al. (2017) control the Lipschitz constant under an ℓ_2 metric by explicitly regularizing the ℓ_2-norm of the gradient, while Lin et al. (2019) essentially control an upper bound on the ℓ_2-norm of the gradient. Comparing against these baselines thus gives insight into how our method of regularizing the ℓ_1-norm of the gradient compares against regularization of the ℓ_2-norm of the gradient.

4.3.1 Setup

Implementation and Complexity Adding the regularization penalty from Equation 4.3 to the training objective requires higher-order gradients. This feature is available in the latest versions of frameworks such as Tensorflow and PyTorch

(of which we have used the latter for all our experiments). Computing $\nabla_{\mathbf{w}} \|\nabla_{\mathbf{w}} \mathcal{L}\|_1$ using automatic differentiation requires $O(2 \times C \times E)$ extra computations, where E is the number of elementary operations in the original forward computation graph, and C is a fixed constant (Baydin et al., 2018). This can be seen from the fact that $\|\nabla_{\mathbf{w}} \mathcal{L}\|_1$ is a function $\mathbb{R}^{|\mathbf{w}|} \rightarrow \mathbb{R}$, where $|\mathbf{w}|$ denotes the number of weights and the computation of the gradient w.r.t. the loss contains E elementary operations, as many as the forward pass. In practice, enabling regularization increased time-per-epoch time on CIFAR10 from 14 seconds to 1:19 minutes for VGG, and from 24 seconds to 3:29 minutes for ResNet-18. On ImageNet epoch-time increased from 33:20 minutes to 4:45 hours for ResNet-18. The training was performed on a single NVIDIA RTX 2080 Ti GPU.

However, in our experiments we observed that it is not necessary to enable regularization from the beginning, as the ℓ_1-norm of the gradients decreases naturally up to a certain point as the training progresses (See Appendix B.4 for more details). We therefore only enable regularization in the last 15 epochs of training or as an additional fine-tuning phase. We experimented with tuning λ_w and λ_y in Equation 4.3 separately but found no benefit. We therefore set $\lambda_w = \lambda_y = \lambda$ for the remainder of this section.

We use a grid-search to find the best setting for λ. Our search criteria is ensuring that the performance of the *unquantized* model is not degraded. In order to choose a sensible range of values we first track the regularization and cross-entropy loss terms and then choose a range of λ that ensures their ratios are in the same order of magnitude. We do not perform any quantization for validation purposes during the training.

Quantization Details We use uniform symmetric quantization (Jacob et al., 2018; Krishnamoorthi, 2018a) in all our experiments unless explicitly specified otherwise. For the CIFAR 10 experiments we fix the activation bit-widths to 4 bits and then vary the weight bits from 8 to 4. For the Imagenet experiments we use the same bit-width for both weights and activations. For the quantization-aware fine-tuning experiments we employ the STE on a fixed (symmetric) quantization grid. In all these experiments we perform a hyperparameter search over learning rates for each of the quantization bit-widths and use a fixed weight decay of $1e-4$. For our experiments with defensive quantization (Lin et al., 2019) we perform

a hyperparameter search over the scaling parameters of the regularizer and the learning rate. We limit the search over the scaling parameters to those mentioned in (Lin et al., 2019) and do not use weight decay. When applying post-training quantization we set the activation ranges using the batch normalization parameters as described in (Nagel et al., 2019b).

When a model is fine-tuned to a target bit-width and evaluated on a higher bit-width, we can trivially represent the original quantized weights and activations by ignoring the higher-order bits, or quantize using the higher bit-width. As using the higher bit-width to quantize shadow weights and activations introduces noise to the model and might yield lower results, we try both approaches and only report a result if quantization using the higher bit-width gives better results.

4.3.2 Effects of Regularization

In order to get a better understanding of our proposed regularizer, we first adopt the visualization method from Hoffman et al. (2019) and illustrate the effects that the quantization in general, and our method in particular, have on the trained classifier's decision boundaries. The result can be seen in Figure 4.3, where we empirically observe that the regularized networks "expands" its decision cells.

Secondly, we investigate in Figure 4.4 the ℓ_1- and ℓ_2-norms of the gradients for all CIFAR-10 test batches on the VGG-like model. We can observe that while the ℓ_2-norms of the gradient are small in the unregularized model, the ℓ_1-norms are orders of magnitude larger. Consequently, when fine-tuning the same model with our method, we see a strong decrease of the ℓ_1-norm.

Finally, we investigate how the predictive distribution of the floating point model, $p(y|\mathbf{x})$, changes when we quantize either an unregularized baseline or a model regularized with our method, thus obtaining $q(y|\mathbf{x})$. We measure this discrepancy using the KL-divergence of the original predictive when using the predictive distribution of the quantized model, i.e. $D_{\mathrm{KL}}(p(y|x)||q(y|x))$, averaged over each test batch. Since our method improves robustness of the loss gradient against small perturbations, we would expect the per-class probabilities to be more robust to perturbations as well, and thus more stable under quantization noise. The result can be seen in Figure 4.5, where we indeed observe that the gap is smaller when quantizing our regularized model.

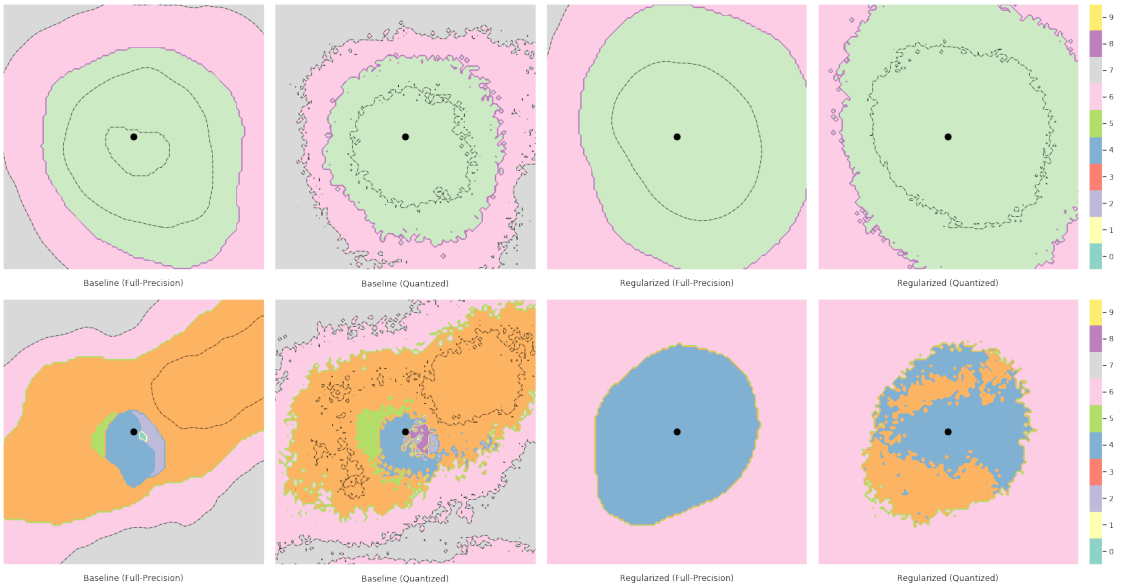


Figure 4.3: Random cross-sections of decision boundaries in the input space. To generate these cross-sections, we draw a random example from the CIFAR-10 test set (represented by the black dot in the center) and pass a random two-dimensional hyper-plane $\subset \mathbb{R}^{1024}$ through it. We then evaluate the network's output for each point on the hyper-plane. Various colors indicate different classes. Softmax's maximum values determine the contours. The top row illustrates the difference between the baseline and the regularized VGG-like networks (and their quantized variants) when they all classify an example correctly. The bottom row depicts a case where the quantized baseline misclassifies an example while the regularized network predicts the correct class. We can see that our regularization pushes the decision boundaries outwards and enlarges the decision cells.

4.3.3 CIFAR-10 and ImageNet Results

The classification results from our CIFAR-10 experiments for the VGG-like and ResNet18 networks are presented in Table 4.1, whereas the result from our ImageNet experiments for the ResNet18 network can be found in Table 4.2. Both tables include all results relevant to the experiment, including results on our method, Defensive Quantization regularization, L2 gradient regularization and finetuning using the STE.

Comparison to "Defensive Quantization" As explained in Section 4.4, Defensive Quantization (Lin et al., 2019) aims to regularize each layer's Lipschitz constant to be close to 1. Since the regularization approach taken by the authors is

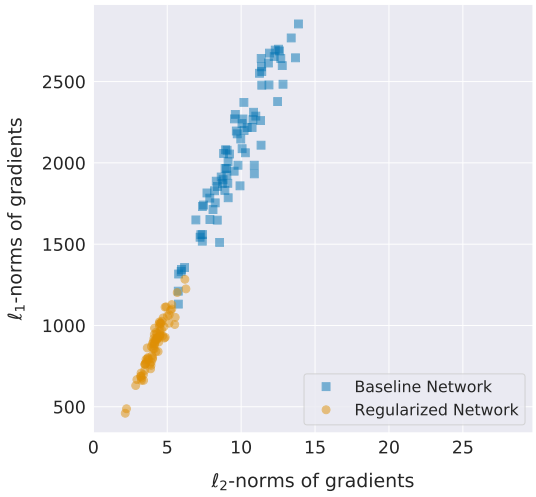


Figure 4.4: ℓ_1- and ℓ_2-norms of the gradients for CIFAR-10 test-set mini-batches. Note the difference between the scales on the horizontal and vertical axis. We observe that our regularization term decreases the ℓ_1-norm significantly, compared to its unregularized counterpart.

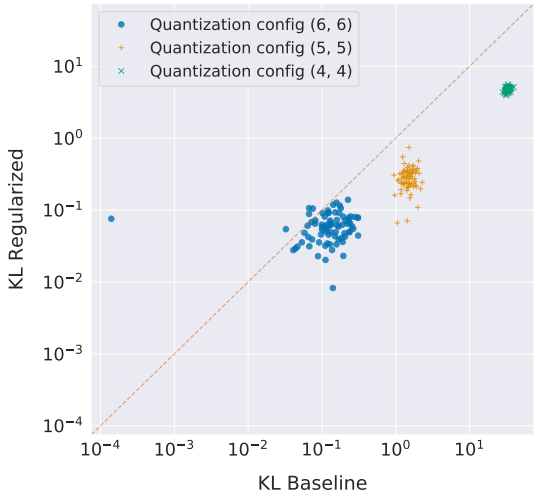


Figure 4.5: KL-divergence of the floating point predictive distribution to the predictive distribution of the quantized model for CIFAR-10 test-set mini-batches. We observe that the regularization leads to a smaller gap, especially for smaller bit-widths.

similar to our method, and the authors suggest that their method can be applied as a regularization for quantization robustness, we compare their method to ours. As the experiments from the original paper differ methodologically from ours in that we quantize both weights and activations, all results on defensive quantization reported in this paper are produced by us. We were able to show improved quantization results using defensive quantization for CIFAR-10 on VGG-like, but not on any of the experiments on ResNet18. We attribute this behavior to too stringent regularization in their approach: the authors regularize *all* singular values of their (reshaped) convolutional weight tensors to be close to one, using a regularization term that is essentially a fourth power regularization of the singular values of the weight tensors (see Appendix B.3). This regularization likely inhibits optimization.

Comparison to explicit ℓ_2-norm gradient regularization We consider the ℓ_2 regularization of the gradient, as proposed by Gulrajani et al. (2017), as a generalization of the DQ regularization. Such regularization has two key benefits over DQ: 1) we can regularize the singular values without reshaping the convolutional

	VGG-like				ResNet-18			
	FP	(8,4)	(6,4)	(4,4)	FP	(8,4)	(6,4)	(4,4)
No Regularization	92.49	79.10	78.84	11.47	93.54	85.51	85.35	83.98
DQ Regularization	91.51	86.30	84.29	30.86	92.46	83.31	83.34	82.47
L2 Regularization	91.88	86.64	86.14	63.93	93.31	84.50	84.99	83.82
L1 Regularization (Ours)	92.63	89.74	89.78	85.99	93.36	88.70	88.45	87.62
STE @ (8,4)	–	91.28	89.99	32.83	–	89.10	87.79	86.21
STE @ (6,4)	–	–	90.25	39.56	–	–	90.77	88.17
STE @ (4,4)	–	–	–	89.79	–	–	–	89.98

Table 4.1: Test accuracy (%) for the VGG-like and ResNet-18 models on CIFAR-10. STE @ (X,X) indicates the weight-activation quantization configuration used with STE for fine-tuning. DQ denotes Defensive Quantization (Lin et al., 2019). For the No Regularization row of results we only report the mean of 5 runs. The full range of the runs is shown in Figure 4.2.

kernels and 2) we impose a less stringent constraint as we avoid enforcing all singular values to be close to one. By observing the results at Table 4.1 and 4.2, we see that the ℓ_2 regularization indeed improves upon DQ. Nevertheless, it provides worse results compared to our ℓ_1 regularization, an effect we can explain by the analysis of Section 4.2.

Comparison to quantization-aware fine-tuning While in general we cannot expect our method to outperform models to which quantization-aware fine-tuning is applied on their target bit-widths, as in this case the model can adapt to that specific quantization noise, we do see that our model performs on par or better when comparing to bit-widths lower than the target bit-width. This is in line with our expectations: the quantization-aware fine-tuned models are only trained to be robust to a specific noise distribution. However, our method ensures first-order robustness regardless of bit-width or quantization scheme, as explained in Section 4.2. The only exception is the 4 bit results on ImageNet. We hypothesize that this is caused by the fact that we tune the regularization strength λ to the highest value that does not hurt full-precision results. While stronger regularization would harm full-precision performance, it would also most likely boost 4 bit results, due to imposing robustness to a larger magnitude, i.e. δ, of quantization noise. Table 4.1 includes results for a higher value of δ that is in line with this analysis.

	Configuration			
	FP	(8,8)	(6,6)	(4,4)
No Regularization	69.70	69.20	63.80	0.30
DQ Regularization	68.28	67.76	62.31	0.24
L2 Regularization	68.34	68.02	64.52	0.19
L1 Regularization (Ours)	70.07	69.92	66.39	0.22
L1 Regularization (Ours) ($\lambda = 0.05$)	64.02	63.76	61.19	55.32
STE @ (8,8)	–	70.06	60.18	0.13
STE @ (6,6)	–	–	69.63	11.34
STE @ (4,4)	–	–	–	57.50

Table 4.2: Test accuracy for the ResNet-18 architecture on ImageNet. STE @ (X,X) indicates the weight-activation quantization configuration used with STE for fine-tuning. In addition to the λ we found through the grid-search which maintains FP accuracy, we also experimented with a stronger $\lambda = 0.05$ to show that (4,4) accuracy can be recovered at the price of overall lower performance.

4.4 Related Work

A closely related line of work to ours is the analysis of the robustness of the predictions made by neural networks subject to an adversarial perturbation in their input. Quantization can be seen as a similar scenario where non-adversarial perturbations are applied to weights and activations instead. Cisse et al. (2017) proposed a method for reducing the network's sensitivity to small perturbations by carefully controlling its global Lipschitz. The Lipschitz constant of a linear layer is equal to the spectral norm of its weight matrix, i.e., its largest singular value. The authors proposed regularizing weight matrices in each layer to be close to orthogonal: $\sum_{\mathbf{W}_l \in \mathbb{W}} \left\| \mathbf{W}_l^T \mathbf{W}_l - \mathbf{I} \right\|^2$. All singular values of orthogonal matrices are one; therefore, the operator does not amplify perturbation (and input) in any direction. Lin et al. (2019) studied the effect of this regularization in the context of quantized networks. The authors demonstrate the extra vulnerability of quantized models to adversarial attacks and show how this regularization, dubbed "Defensive Quantization", improves the robustness of quantized networks.

The idea of regularizing the norm of the gradients has been proposed before (Gulrajani et al., 2017) in the context of GANs, as another way to enforce Lipschitz continuity. This approach has a major advantage over the methods mentioned above.

Using weight regularization is only well-defined for 2D weight matrices such as in fully-connected layers. The penalty term is often approximated for convolutional layers by reshaping the weight kernels into 2D matrices. Sedghi et al. (2018) showed that the singular values found in this weight could be very different from the actual operator norm of the convolution. Some operators, such as nonlinearities, are also ignored. Regularizing Lipschitz constant through gradients does not suffer from these shortcomings, and the operator-norm is regularized directly.

Guo et al. (2018) demonstrated that there exists an intrinsic relationship between sparsity in DNNs and their robustness against ℓ_∞ and ℓ_2 attacks. For a binary linear classifier, the authors showed that they could control the ℓ_∞ robustness, and its relationship with sparsity, by regularizing the ℓ_1 norm of the weight tensors. In the case of a linear classifier, this objective is, in fact, equivalent to our proposed regularization penalty.

Finally, another line of work related to ours revolves around quantization-aware training. This can, in general, be realized in two ways: 1) regularization and 2) mimicking the quantization procedure during the forward pass of the model. In the first case, we have methods (Yin et al., 2018; Achterhold et al., 2018) where there are auxiliary terms introduced in the objective function such that the optimized weights are encouraged to be near, under some metric, to the quantization grid points, thus alleviating quantization noise. In the second case, we have methods that rely on either the STE (Courbariaux et al., 2015; Rastegari et al., 2016; Jacob et al., 2018), stochastic rounding (Gupta et al., 2015; Gysel, 2016), or surrogate objectives and gradients (Louizos et al., 2018; Shayer et al., 2017). While all of the methods above have been effective, they still suffer from a major limitation; they target one-specific bit-width. In this way, they are not appropriate for use-cases where we want to be able to choose the bit-width "on the fly".

4.5 Summary

In this chapter, we analyzed the effects of the quantization noise on the loss function of neural networks. By modelling quantization as an ℓ_∞-bounded perturbation, we showed how we can control the first-order term of the Taylor expansion of the loss by a straightforward regularizer that encourages the ℓ_1-norm of the gradients to be

small. We empirically confirmed its effectiveness, demonstrating that standard post-training quantization to such regularized networks can maintain good performance under a variety of settings for the bit-width of the weights and activations. As a result, our method paves the way towards quantizing floating-point models "on the fly" according to bit-widths that are appropriate for the resources currently available.

Chapter 5

Data Compressing Using Meta-Gradients

In the previous two chapters we explored how the information in meta-gradients can be used for compressing neural networks with pruning and quantization. In this chapter we look at the dual compression problem, i.e. compressing data using neural networks, also known as neural compression.

Neural compression methods are typically based on autoencoders and heavily rely on encoder/decoder architectures that are specialized for images. As a result, most of these methods are not effective for non-image data sources (Ballé et al., 2018a; Minnen et al., 2018a; Lee et al., 2019a).

In this chapter we first propose a simple autoencoder-free approach for image compression using Implicit Neural Representations. We then propose an improved version of this method and show how meta-gradients can be used to learn a base network for quick data compression. We show that this method is applicable to a wide range of data modalities, from images and audio to medical and climate data.

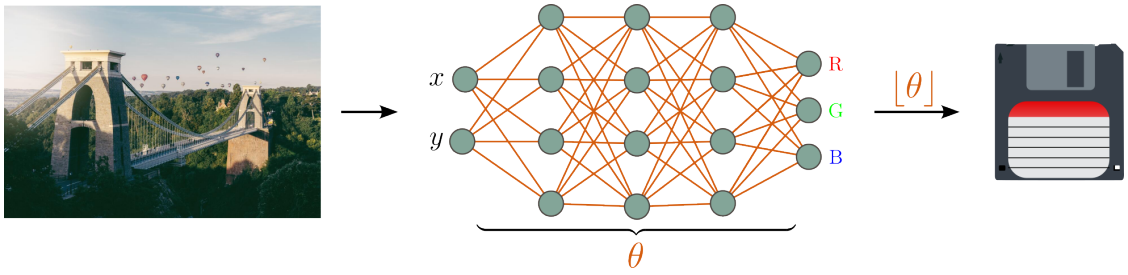


Figure 5.1: Compressed implicit neural representations (COIN). We overfit an image with a neural network mapping pixel locations (x, y) to RGB values (often referred to as an implicit neural representation). We then quantize the weights of this neural network to a lower bit-width and transmit them.

5.1 Compression Using Implicit Neural Representations

We start with image compression and take a different approach: we encode an image by overfitting it with a small MLP mapping pixel locations to RGB values and then transmit the weights θ of this MLP as a code for the image. Figure 5.1 visualizes our approach. While overfitting such MLPs is difficult due to the high-frequency information contained in natural images (Basri et al., 2019; Tancik et al., 2020b), recent research has shown that this can be mitigated by using sinusoidal encodings and activations (Mildenhall et al., 2020; Tancik et al., 2020b; Sitzmann et al., 2020b). We show that using MLPs with sine activations, often referred to as SIRENs (Sitzmann et al., 2020b), we can fit large images (393k pixels) with surprisingly small networks (8k parameters).

We now describe our proposed method for image compression COmpressed Implicit Neural representations (COIN) in more details:

Encoding. The encoding step consists in overfitting an MLP to the input data, quantizing its weights and transmitting these. We consider compressing data that can be expressed in terms of sets of coordinates $\mathbf{x} \in \mathcal{X}$ and features $\mathbf{y} \in \mathcal{Y}$. An image for example can be described by a set of pixel locations $\mathbf{x} = (x, y)$ in $\mathbb{R}^2$ and their corresponding RGB values $\mathbf{y} = (r, g, b)$ in $\{0, 1, ..., 255\}^3$. Similarly, an MRI scan can be described by a set of positions in 3D space $\mathbf{x} = (x, y, z)$ and an intensity value $\mathbf{y} \in \mathbb{R}^+$. Given a single datapoint as a collection of coordinate and

feature pairs $\mathbf{d} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (for example an image as a collection of n pixel locations and RGB values), the COIN approach consists in fitting a neural network $f_\theta : \mathcal{X} \to \mathcal{Y}$ with parameters θ to the datapoint by minimizing the mean squared error (MSE)

$$\mathcal{L}(\theta, \mathbf{d}) = \sum_{i=1}^{n} \|f_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2. \tag{5.1}$$

The weights θ are then quantized and stored as a compressed representation of the datapoint $\mathbf{d}$. The neural network f_θ is parameterized by a SIREN (Sitzmann et al., 2020b), i.e. an MLP with sine activation functions, which is necessary to fit high frequency data such as natural images (Mildenhall et al., 2020; Tancik et al., 2020b; Sitzmann et al., 2020b). More specifically, a SIREN layer is defined by an elementwise sin applied to a hidden feature vector $\mathbf{h} \in \mathbb{R}^d$ as

$$\text{SIREN}(\mathbf{h}) = \sin(\omega_0(W\mathbf{h} + \mathbf{b})) \tag{5.2}$$

where $W \in \mathbb{R}^{d \times d}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^d$ a bias vector and $\omega_0 \in \mathbb{R}^+$ a positive scaling factor.

Decoding. Given the stored quantized weights θ, decoding simply consists in evaluating the function f_θ at every pixel location to reconstruct the image. This decoding approach gives us extra flexibility: we can progressively decode the image, e.g. by decoding parts of the image or a low resolution image first, simply by evaluating the function at various pixel locations. Partially decoding images in this way is difficult with autoencoder based methods, showing a further advantage of the COIN approach.

Limitations. The main limitation of our approach is that encoding is slow, because we have to solve an optimization problem for each encoded image. However, paying a significant computational cost upfront to compress content for delivery to many receivers is a standard practice in the setting of one-to-many media distribution, e.g., at Netflix (Aaron et al., 2015). Further, at decoding time, we are required to evaluate the network at every pixel location to decode the full image. However, this computation can be embarrassingly parallelized to the point of a single forward pass for all pixels. Finally, our method performs worse than state-of-the-art compression methods. However, there are several promising directions to reduce this gap.

5.2 Compression Using Meta-Gradients and Modulations

While this COIN approach introduced in the previous section is very general, there are several key issues. Firstly, as compression involves minimizing equation 5.1, encoding is extremely slow. For example, compressing a single image from the Kodak dataset (Kodak, 1991) takes nearly an hour on a 1080Ti GPU (Dupont et al., 2021a). Secondly, as each datapoint $\mathbf{d}$ is fitted with a separate neural network f_θ, there is no information shared across data points. This is clearly suboptimal; natural images for example share a lot of common structure that does not need to be repeatedly stored for each individual image.

In the following sections, we propose an improved compression algorithm using meta-gradients, which we call COIN++, that addresses these issues while maintaining the generality and applicability to multiple modalities. We address these issues by:

1. Using meta-gradients and meta-learning to reduce encoding time by more than two orders of magnitude to less than a second, compared to minutes or hours for COIN (Sitzmann et al., 2020a; Tancik et al., 2020a),
2. Learning a base network that encodes shared structure and applying modulations to this network to encode instance specific information,
3. Quantizing and entropy coding the modulations.

While our method significantly exceeds COIN both in terms of compression and speed, it only partially closes the gap to SOTA codecs on well-studied modalities such as images. However, COIN++ is applicable to a wide range of data modalities where traditional methods cannot be used, making it a promising tool for neural compression in non-standard domains. Figure 5.2 illustrates the COIN++ architecture.

5.2.1 Storing Modulations

While COIN stores each image as a separate neural network, we instead train a base network shared across datapoints and apply *modulations* to this network to

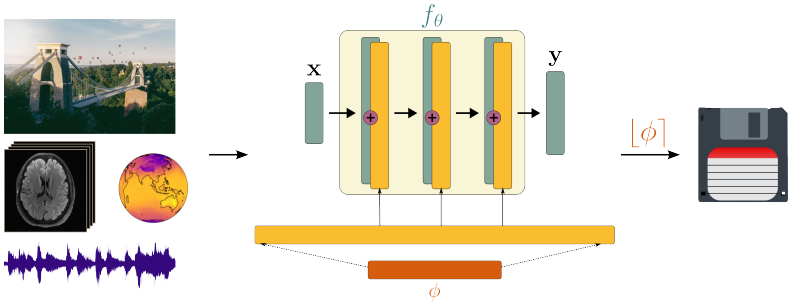


Figure 5.2: COIN++ framework can convert a wide range of data modalities to neural networks via optimization and then storing the parameters of these neural networks as compressed codes for the data. COIN++ architecture: Latent modulations ϕ are mapped to modulations which are added to activations of the base network f_θ to parameterize a single function that can be evaluated at coordinates $\mathbf{x}$ to obtain features $\mathbf{y}$.

parameterize individual datapoints. Given a base network, such as a multi-layer perceptron (MLP), we use FiLM layers (Perez et al., 2018), to modulate the hidden features $\mathbf{h} \in \mathbb{R}^d$ of the network by applying elementwise scales $\boldsymbol{\gamma} \in \mathbb{R}^d$ and shifts $\boldsymbol{\beta} \in \mathbb{R}^d$ as

$$\text{FiLM}(\mathbf{h}) = \boldsymbol{\gamma} \odot \mathbf{h} + \boldsymbol{\beta}. \tag{5.3}$$

Given a fixed base MLP, we can therefore parameterize families of neural networks by applying different scales and shifts at each layer. Each neural network function is therefore specified by a set of scales and shifts, which are collectively referred to as modulations (Perez et al., 2018). Recently, the FiLM approach has also been applied in the context of INRs. Chan et al. (2021) parameterize the generator in a generative adversarial network by a SIREN network and generate samples by applying modulations to this network as $\sin(\boldsymbol{\gamma} \odot (W\mathbf{h} + \mathbf{b}) + \boldsymbol{\beta})$. Similarly, Mehta et al. (2021) parameterize families of INRs using a scale factor via $\boldsymbol{\alpha} \odot \sin(W\mathbf{h} + \mathbf{b})$. Both of these approaches can be modified to use a low dimensional latent vector mapped to a set of modulations instead of directly applying modulations. Chan et al. (2021) map a latent vector to scales and shifts with an MLP, while Mehta et al. (2021) map the latent vector through an MLP of the same shape as the base network and use the hidden activations of this network as modulations.

We use a similar approach for COIN++. Instead of storing the weights of a neural network for each datapoint, we store a set of modulations applied to a shared base network. More specifically, given a base SIREN network, we only apply shifts

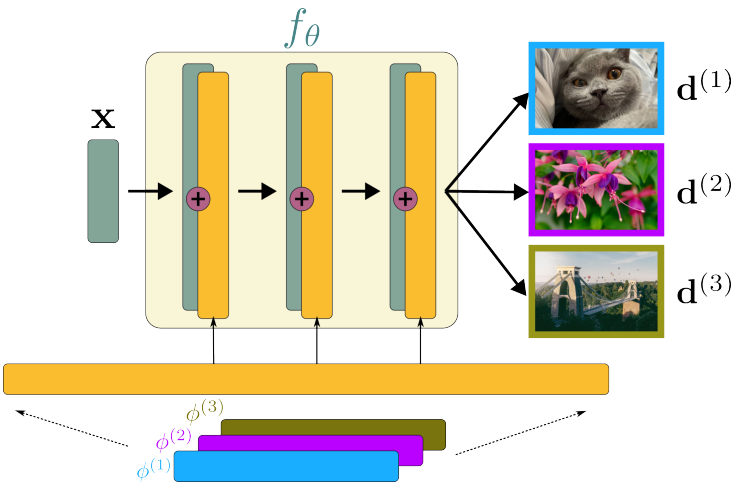


Figure 5.3: By applying modulations $\phi^{(1)}$, $\phi^{(2)}$, $\phi^{(3)}$ to a base network f_θ, we obtain different functions that can be decoded into datapoints $\mathbf{d}^{(1)}$, $\mathbf{d}^{(2)}$, $\mathbf{d}^{(3)}$ by evaluating the functions at various coordinates. While we show images in this figure, the same principle can be applied to a range of data modalities.

$\boldsymbol{\beta} \in \mathbb{R}^d$ as modulations using

$$\sin(\omega_0(W\mathbf{h} + \mathbf{b} + \boldsymbol{\beta})) \tag{5.4}$$

at every layer of the MLP. Indeed, we found empirically that using only shifts gave the same performance as using both shifts and scales while only requiring half the modulations and hence half the storage. Using only scales yielded considerably worse performance. To further reduce storage, we use a latent vector which is *linearly* mapped to the modulations as shown in Figure 5.2[1]. In a slight overload of notation, we also refer to this vector as modulations or latent modulations. We then store a datapoint $\mathbf{d}$ (such as an image) as a set of (latent) modulations ϕ. To decode the datapoint, we simply evaluate the modulated base network $f_\theta(\cdot; \phi)$ at every coordinate $\mathbf{x}$,

$$\mathbf{y} = f_\theta(\mathbf{x}; \phi) \tag{5.5}$$

as shown in Figure 5.3. To fit a set of modulations ϕ to a datapoint $\mathbf{d}$, we keep the parameters θ of the base network fixed and minimize

[1]We found that our parameterization (shifts with a linear map) performed significantly better than the parameterizations by Chan et al. (2021) and Mehta et al. (2021).

$$\mathcal{L}(\theta, \phi, \mathbf{d}) = \sum_{i=1}^{n} \| f_\theta(\mathbf{x}_i; \phi) - \mathbf{y}_i \|_2 \tag{5.6}$$

over ϕ. In contrast to COIN, where each datapoint $\mathbf{d}$ is stored as a separate neural network f_θ, COIN++ only requires storing $O(n)$ modulations (or less when using latents) as opposed to $O(n^2)$ weights, where n is the width of the MLP. In addition, this approach allows us to store shared information in the base network and instance specific information in the modulations. For natural images for example, the base network encodes structure that is common to natural images while the modulations store the information required to reconstruct individual images.

5.2.2 Meta-Learning Modulations

Given a base network f_θ, we can encode a datapoint $\mathbf{d}$ by minimizing equation 5.6. However, we are still faced with two problems: *1.* We need to learn the weights θ of the base network, *2.* Encoding a datapoint via equation 5.6 is slow, requiring thousands of iterations of gradient descent. COIN++ solves both of these problems with meta-learning.

Recently, Sitzmann et al. (2020a) and Tancik et al. (2020a) have shown that applying MAML (Finn et al., 2017b) to INRs can reduce fitting at test time to just a few gradient steps. Instead of minimizing $\mathcal{L}(\theta, \mathbf{d})$ directly via gradient descent from a random initialization, we can meta-learn an initialization θ^* such that minimizing $\mathcal{L}(\theta, \mathbf{d})$ can be done in a few gradient steps. More specifically, assume we are given a dataset of N points $\{\mathbf{d}^{(j)}\}_{j=1}^N$. Starting from an initialization θ, a step of the MAML inner loop on a datapoint $\mathbf{d}^{(j)}$ is given by

$$\theta^{(j)} = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathbf{d}^{(j)}), \tag{5.7}$$

where α is the inner loop learning rate. We are then interested in learning a good initialization θ^* such that the loss $\mathcal{L}(\theta, \mathbf{d}^{(j)})$ is minimized after a few gradient steps across the entire set of datapoints $\{\mathbf{d}^{(j)}\}_{j=1}^N$. To update the initalization θ, we then perform a step of the outer loop, with an outer loop learning rate β, via

$$\theta \leftarrow \theta - \beta \nabla_\theta \textstyle\sum_{j=1}^N \mathcal{L}(\theta^{(j)}, \mathbf{d}^{(j)}). \tag{5.8}$$

In our case, MAML cannot be used directly since at test time we only fit the

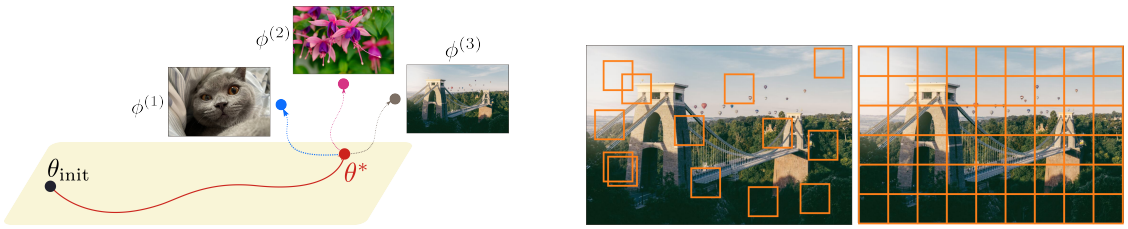


Figure 5.4: (Left) We meta-learn parameters θ^* of the base network such that modulations ϕ can easily be fit in a few gradient steps. (Right) During training we sample patches randomly, while at test time we partition the datapoint into patches and fit modulations to each patch.

modulations ϕ and not the shared parameters θ. We therefore need to meta-learn an initialization for θ and ϕ such that, given a new datapoint, the *modulations* ϕ can rapidly be computed while keeping θ constant. Indeed, we only store the modulations for each datapoint and share the parameters θ across all datapoints. For COIN++, a single step of the inner loop is then given by

$$\phi^{(j)} = \phi - \alpha \nabla_{\phi} \mathcal{L}(\theta, \phi, \mathbf{d}^{(j)}), \tag{5.9}$$

where θ is kept fixed. Performing the inner loop on a subset of parameters has previously been explored by Zintgraf et al. (2019a) and is referred to as CAVIA. As observed in CAVIA, meta-learning the initialization for ϕ is redundant as it can be absorbed into a bias parameter of the base network weights θ. We therefore only need to meta-learn the shared parameter initialization θ. The update rule for the outer loop is then given by

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \textstyle\sum_{j=1}^{N} \mathcal{L}(\theta, \phi^{(j)}, \mathbf{d}^{(j)}). \tag{5.10}$$

The inner loop then updates the modulations ϕ while the outer loop updates the shared parameters θ. This algorithm allows us to meta-learn a base network such that each set of modulations can easily and rapidly be fitted (see Figure 5.4). In practice, we find that as few as 3 gradient steps gives us compelling results, compared with thousands for COIN.

5.2.3 Patches, Quantization, and Entropy Coding

Patches for large scale data. While meta-learning the base network allows us to rapidly encode new datapoints into modulations, the training procedure is expensive, as MAML must take gradients through the inner loop (Finn et al., 2017b). For large datapoints (such as high resolution images or MRI scans), this can become prohibitively expensive. While first-order approximations exist (Finn et al., 2017b; Nichol et al., 2018; Rajeswaran et al., 2019), we found that they severely hindered performance. Instead, to reduce memory usage, we split datapoints into random patches during training. For large scale images for example, we train on 32×32 patches. At train time, we then learn a base network such that modulations can easily be fit to patches. At test time, we split a new image into patches and compute modulations for each of them. The image is then represented by the set of modulations for all patches (see Figure 5.4). We use a similar approach for other data modalities, e.g. MRI scans are split into 3D patches.

Quantization. While COIN quantizes the neural network weights from 32 bits to 16 bits to reduce storage, quantizing beyond this severely hinders performance (Dupont et al., 2021a). In contrast, we find that modulations are surprisingly quantizable. During meta-learning, modulations are represented by 32 bit floats. To quantize these to shorter bitwidths, we simply use uniform quantization. We first clip the modulations to lie within 3 standard deviations of their mean. We then split this interval into 2^b equally sized bins (where b is the number of bits). Remarkably, we found that reducing the number of bits from 32 to 5 (i.e. reducing the number of symbols from more than 10^9 to only 32) resulted only in small decreases in reconstruction accuracy. Simply applying uniform quantization then improves compression by a factor of 6 at little cost in reconstruction quality.

Entropy coding. A core component of almost all codecs is entropy coding, which allows for lossless compression of the quantized code, using e.g. arithmetic coding (Rissanen et al., 1979). This relies on a model of the distribution of the quantized codes. As with quantization, we use a very simple approach for modeling this distribution: we count the frequency of each quantized modulation value and use this distribution for arithmetic coding. In our experiments, this reduced storage 8-15% at no cost in reconstruction quality. While this simple entropy coding

scheme works well, we expect more sophisticated methods to significantly improve performance, which is an exciting direction for future work.

5.2.4 Limitations

The main drawback of our method is that computing meta-gradients is memory intensive. This in turn limits scalability and requires us to use patches for large data. In addition, training COIN++ can occasionally be unstable, although the model typically recovers from loss instabilities (see Figure C.10 in the appendix). Further, there are several common modalities our framework cannot handle, such as text or tabular data, as these are not easily expressible as continuous functions. While out methods do not match the performance of SOTA codecs, we hope this work will lead a novel class of methods for neural data compression and help expand the range of domains where neural compression is applicable.

5.3 Experimental Results

In this section we empirically evaluate our methods COIN and COIN++ across a wide range of data modalities, from images and audio to medical and climate data.

5.3.1 COIN

We perform experiments on the Kodak image dataset (Kodak, 1991) consisting of 24 images of size 768×512. We compare our model against three autoencoder based neural compression baselines which we refer to as BMS (Ballé et al., 2018a), MBT (Minnen et al., 2018a), and CST (Cheng et al., 2020). We also compare against the JPEG, JPEG2000, BPG and VTM image codecs. To benchmark our model, we use the CompressAI library (Bégaint et al., 2020) and the pre-trained models provided therein. We implement our model in PyTorch (Paszke et al., 2019b) and perform all experiments on a single RTX 2080 Ti GPU.

Rate-distortion plots. To determine the best model architectures for a given parameter budget (measured in bits per pixel or bpp[2]), we first find valid combina-

[2] bits-per-pixel $= \frac{\#\text{parameters} \times \text{bits-per-parameter}}{\#\text{pixels}}$

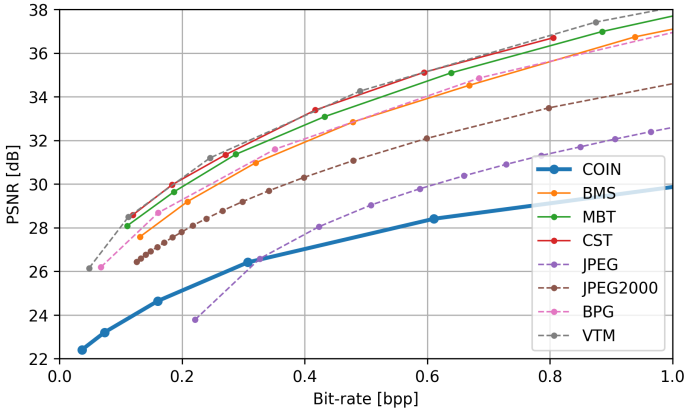


Figure 5.5: COIN rate-distortion on KODAK

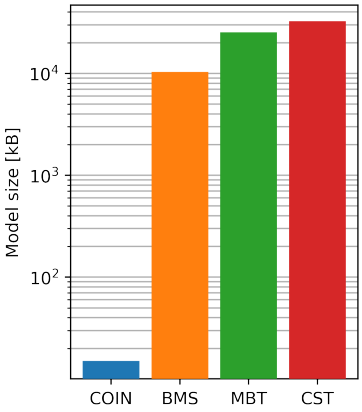


Figure 5.6: COIN model sizes for KODAK (0.3bpp)

tions of depth and width for the MLPs representing an image. For example, for 0.3bpp using 16-bit weights, valid networks include MLPs with 10 layers of width 28, 7 layers of width 34 and so on. We then select the best architecture by running a hyperparameter search over learning rates and valid architectures on a single image using Bayesian optimization (we found that the results of the architecture search transferred well to other images). The resulting model is trained on each image in the dataset at 32-bit precision and converted to 16-bit precision after training. We note that decreasing the weights' precision from 32-bit to 16-bit after training resulted in almost no distortion increase, but decreasing them further to 8-bit incurred significant amount of distortion, outweighing the benefit of halving the bpp.

Results of this procedure for various bpp levels are shown in Figure 5.5. As can be seen, at low bit-rates our model improves upon JPEG *even without using entropy coding*. While our approach is still far from the state of the art compression methods, we believe the performance of such a simple approach is promising for future work in this direction.

Model size. In contrast to most other neural data compression algorithms, our method does not require a decoder at test time. Indeed, while the latent code representing the compressed image in such methods is small, the decoder model is large (typically much larger than an uncompressed image). As such, the memory required on the decoding device is also large. In our case, we only require the

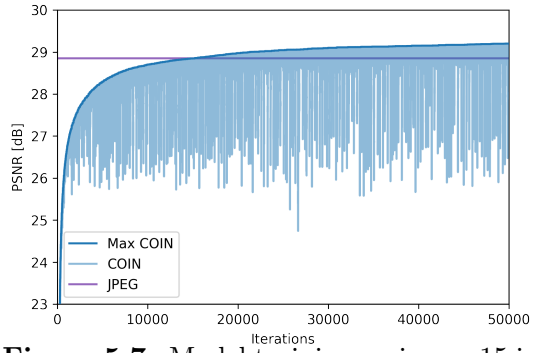


Figure 5.7: Model training on image 15 in the Kodak dataset. Max COIN represents the max PSNR achieved at any point during training.

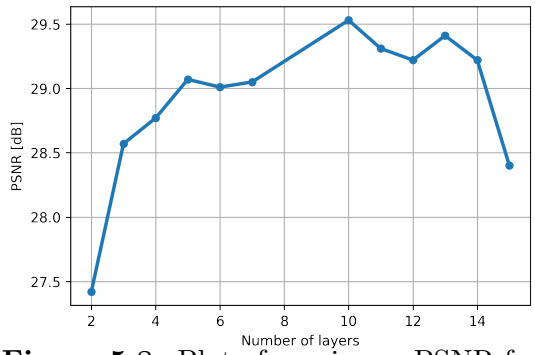


Figure 5.8: Plot of maximum PSNR for networks of the same size (0.3bpp) with different architectures.

weights of a (very small) MLP on the decoder side, leading to memory requirements that are orders of magnitude smaller. As can be seen in Figure 5.6, at 0.3bpp, our method requires 14kB, whereas other baselines require between 10MB and 40MB.

Encoding optimization dynamics. We show an example of the overfitting procedure in Figure 5.7. As can be seen, COIN outperforms JPEG after 15k iterations and continues improving beyond that. While the optimization can be noisy, we simply save the model with the best PSNR.

Architecture choice. In Figure 5.8, we show the performance of various valid architectures of size 0.3bpp. As can be seen, the quality of compression depends on the architecture choice, with different optimal architectures for different bpp values, see Appendix C for details.

5.3.2 COIN++

We evaluate COIN++ on four data modalities: images, audio, medical data and climate data. We implement all models in PyTorch (Paszke et al., 2019b) and train on a single GPU. We use SGD for the inner loop with a learning rate of 1e-2 and Adam for the outer loop with a learning rate of 1e-6 or 3e-6. We normalize coordinates $\mathbf{x}$ to lie in $[-1, 1]$ and features $\mathbf{y}$ to lie in $[0, 1]$. Full experimental details required to reproduce all the results can be found in the Appendix C. We train COIN++ using MSE between the compressed and ground truth data. As is standard, we measure reconstruction performance (or distortion) using PSNR

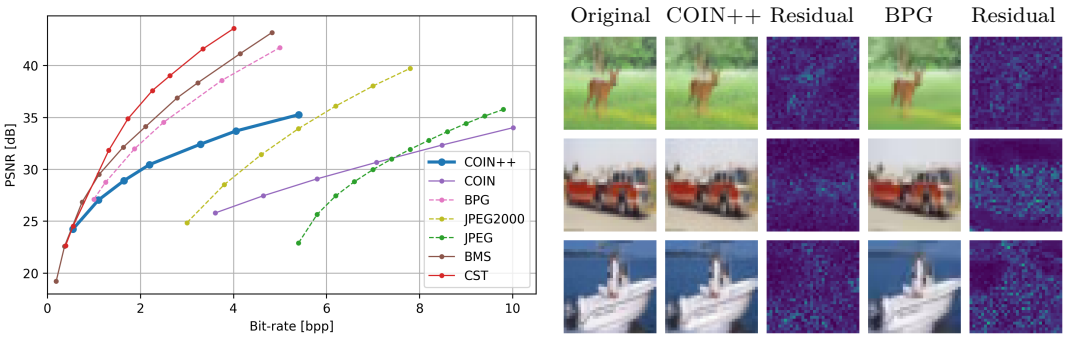


Figure 5.9: (Left) Rate distortion plot on CIFAR10. (Right) Qualitative comparison of compression artifacts for models at similar reconstruction quality. COIN++ achieves 32.4dB at 3.29 bpp while BPG achieves 31.9dB at 1.88 bpp.

(in dB), which is defined as PSNR $= -10 \log_{10}(\text{MSE})$. We measure the size of the compressed data (or rate) in terms of bits-per-pixel (bpp) which is given by $\frac{\text{number of bits}}{\text{number of pixels}}$ [3] and kilobits per second (kpbs) for audio. We benchmark COIN++ against a large number of baselines including standard image codecs - JPEG (Wallace, 1992), JPEG2000 (Skodras et al., 2001), BPG (Bellard, 2014) and VTM (Bross et al., 2021) - autoencoder-based neural compression - BMS (Ballé et al., 2018a), MBT (Minnen et al., 2018a) and CST (Cheng et al., 2020) - standard audio codecs - MP3 (MP3, 1993) - and COIN (Dupont et al., 2021a). For clarity, we use consistent colors for different codecs and plot learned codecs with solid lines and standard codecs with dashed lines. The code to reproduce all experiments in the paper is available online [4].

Images: CIFAR10

We train COIN++ on CIFAR10 using 128, 256, 384, 512, 768 and 1024 latent modulations. As can be seen in Figure 5.9, COIN++ vastly outperforms COIN, JPEG and JPEG2000 while partially closing the gap to BPG, particularly at low bitrates. To the best of our knowledge, this is the first time compression with INRs has outperformed image codecs like JPEG2000. The remaining gap between COIN++ and SOTA codecs (BMS, CST) is likely due to entropy coding: we use the simple scheme described in Section 5.2.3, while BMS and CST use deep generative models. We hypothesize that using deep entropy coding for the modulations would

[3] For non image data a "pixel" corresponds to a single dimension of the data.
[4] https://github.com/EmilienDupont/coinpp

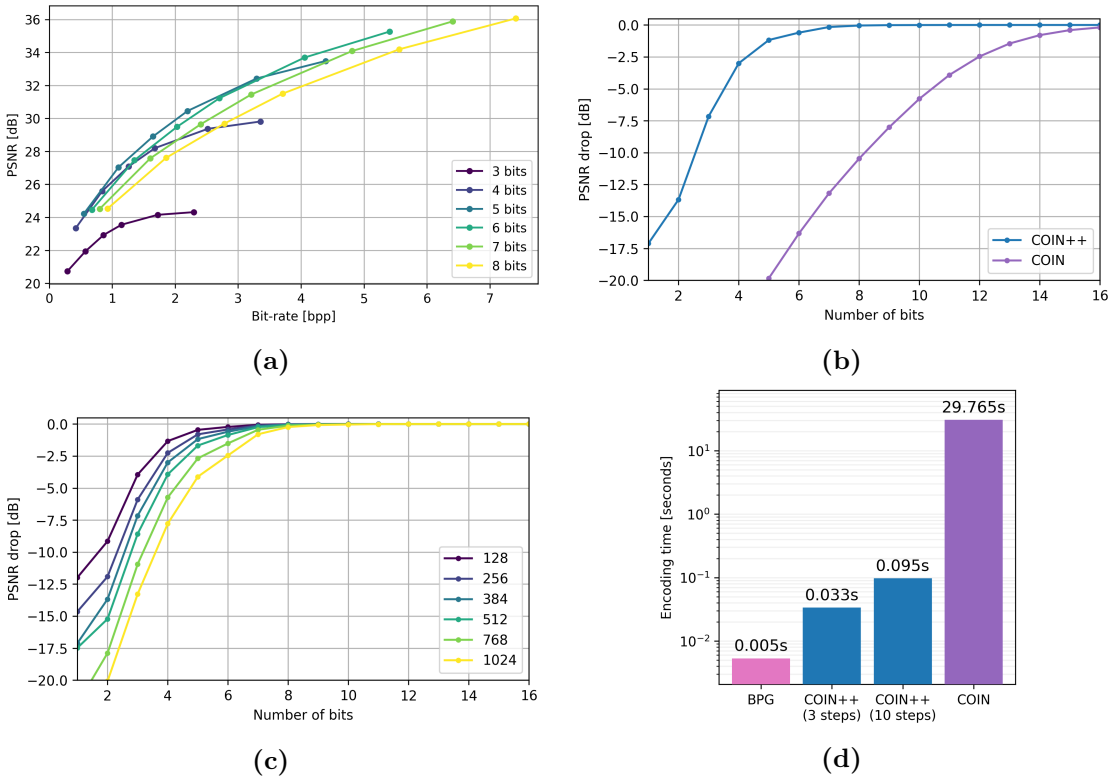


Figure 5.10: (a) Rate distortion plot on CIFAR10 when quantizing the modulations ϕ to various bitwidths. (b) Drop in PSNR for COIN and COIN++ quantization. (c) Drop in PSNR when quantizing the modulations ϕ to various bitwidths, for various latent dimensions. (d) Encoding time per image on CIFAR10 (log scale).

significantly reduce or close this gap. Figure 5.9 shows qualitative comparisons between our model and BPG to highlight the types of compression artifacts obtained with COIN++. In order to thoroughly analyse and evaluate each component of COIN++, we perform a number of ablation studies.

Quantization bitwidth. Quantizing the modulations to a lower bitwidth yields more compressed codes at the cost of reconstruction accuracy. To understand the tradeoff between these, we show rate distortion plots when quantizing from 3 to 8 bits in Figure 5.10a. As can be seen, the optimal bitwidths are surprisingly low: 5 bits is optimal at low bitrates while 6 is optimal at higher bitrates. Qualitative artifacts obtained from quantizing the modulations are shown in Figure C.13 in the appendix.

Quantization COIN vs COIN++. We compare the drop in PSNR due

to quantization for COIN and COIN++ in Figure 5.10b. As can be seen, modulations are remarkably quantizable: when quantizing the COIN weights directly, performance decreases significantly around 14 bits, whereas quantizing modulations yields small drops in PSNR even when using 5 bits. However, as shown in Figure 5.10c, the drop in PSNR from quantization is larger for larger models.

Entropy coding. Figure C.11 in the appendix shows rate distortion plots for full precision, quantized and entropy coded modulations. As can be seen, both quantization and entropy coding significantly improve performance.

Encoding time. Figure 5.10d shows the average encoding time for COIN++, COIN and BPG on CIFAR10 (see appendix C.3.1 for hardware details). As can be seen, COIN++ compresses images 300× faster than COIN while achieving a 4× better compression rate. Note that these results are obtained from compressing each image separately. When using batches of images, we can compress the entire CIFAR10 test set (10k images) in 4mins when using 10 inner loop steps (and in just over a minute when using 3 steps). In addition, as shown in Figure C.12 in the appendix, COIN++ requires only 3 gradient steps to reach the same performance as COIN does in 10,000 steps, while using 4× less storage.

Climate data: ERA5 global temperature measurements

To demonstrate the flexibility of our approach, we also use COIN++ to compress data lying on a manifold. We use global temperature measurements from the ERA5 dataset (Hersbach et al., 2019) with the processing and splits from Dupont et al. (2021b). The dataset contains 8510 train and 2420 test globes of size 46×90, with temperature measurements at equally spaced latitudes λ and longitudes φ on the Earth from 1979 to 2020. To model this data, we follow Dupont et al. (2021b) and use spherical coordinates $\mathbf{x} = (\cos\lambda\cos\varphi, \cos\lambda\sin\varphi, \sin\lambda)$ for the inputs. As a baseline, we compare COIN++ against JPEG, JPEG2000, and BPG applied to flat map projections of the data. As can be seen in Figure 5.11, COIN++ vastly outperforms all baselines. These strong results highlight the versatility of the COIN++ approach: unlike traditional codecs and autoencoder based methods (which would require spherical convolutions for the encoder), we can easily apply our method to a wide range of data modalities, including data lying on a manifold.

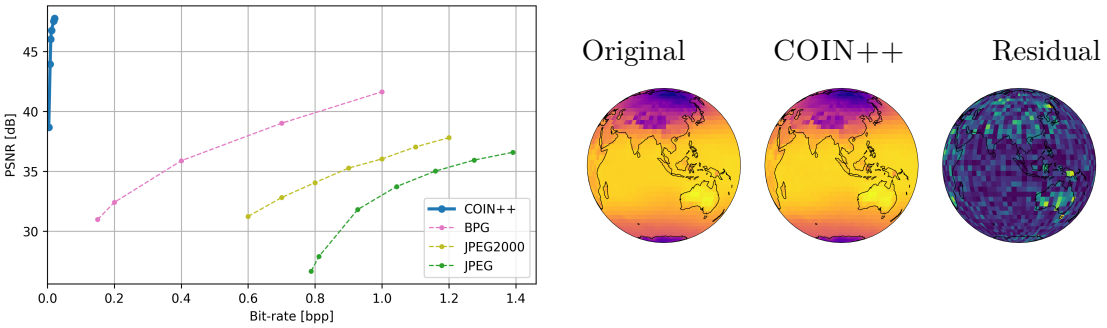


Figure 5.11: (Left) Rate distortion plot on ERA5. (Right) COIN++ compression artifacts on ERA5. See appendix C.12 for more samples.

Indeed, COIN++ achieves a 3000× compression rate while having an RMSE of 0.5°C, highlighting the potential for compressing climate data.

Compression with patches

To evaluate the patching approach from Section 5.2.3 and to demonstrate that COIN++ can scale to large data (albeit at a cost in performance), we test our model on images, audio and MRI data.

Large scale images: Kodak. The Kodak dataset (Kodak, 1991) contains 24 large scale images of size 768×512. To train the model, we use random 32×32 patches from the Vimeo90k dataset (Xue et al., 2019), containing 154k images of size 448×256. At evaluation time, each Kodak image is then split into 384 32×32 patches which are compressed independently. As we do not model the global structure of the image, we therefore expect a significant drop in performance compared to the case when no patching is required. As can be seen in Figure 5.12, the performance of COIN++ indeed drops, but still outperforms COIN and JPEG at low bitrates. We expect that this can be massively improved by modeling the global structure of the image (e.g. two patches of blue sky are nearly identical, but that information redundancy is not exploited in the current setup) but leave this to future work.

Audio: LibriSpeech. To evaluate COIN++ on audio, we use the LibriSpeech dataset (Panayotov et al., 2015) containing several hours of speech data recorded at 16kHz. As a baseline, we compare against the widely used MP3 codec (MP3, 1993). We split each audio sample into patches of varying size and compress each

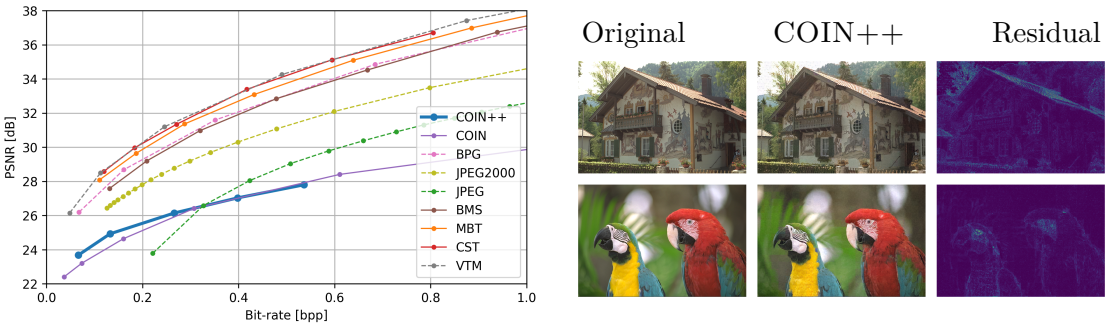


Figure 5.12: (Left) Rate distortion plot on KODAK. (Right) COIN++ compression artifacts on KODAK. See appendix C.12 for more samples.

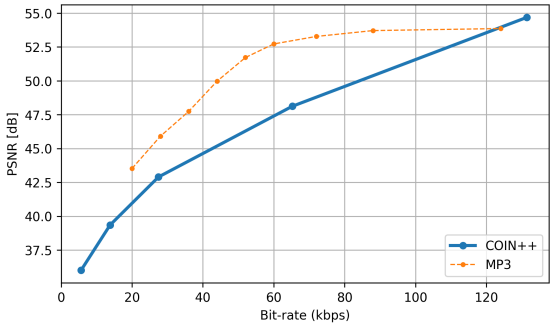


Figure 5.13: Rate distortion plot on LibriSpeech.

of these to obtain models at various bit-rates (we refer to appendix C.3.5 for full experimental details). As can be seen in Figure 5.13, even though audio is a very different modality from the rest considered in this paper, COIN++ can still be used for compression, highlighting the versatility of our approach. However, in terms of performance, COIN++ still lags behind well-establised audio codecs such as MP3.

Medical data: brain MRI scans. Finally, we train our model on brain MRI scans from the FastMRI dataset (Zbontar et al., 2018). The dataset contains 565 train volumes and 212 test volumes with sizes ranging from 16×320×240 to 16×384×384 (see appendix C.1.2 for full dataset details). As a baseline, we compare our model against JPEG, JPEG2000 and BPG applied independently to each slice. Due to memory constraints, we train COIN++ on 16×16×16 patches. We therefore store roughly 400 independent patches at test time (as opposed to 16 slices for the image codecs). Even then COIN++ performs reasonably well, particularly at low bitrates

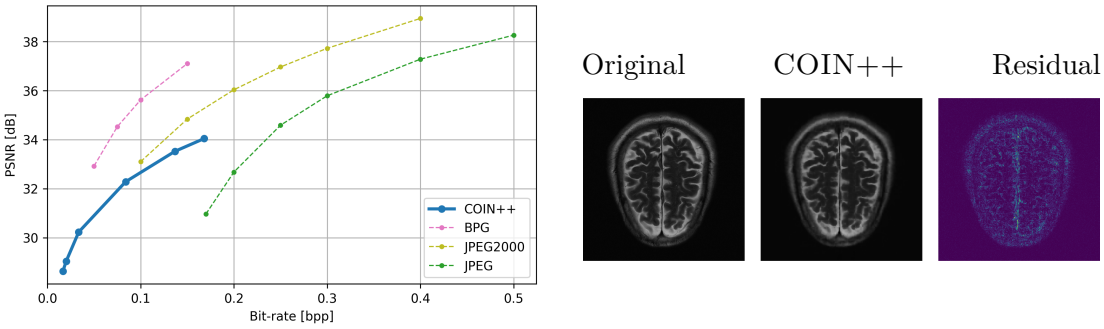


Figure 5.14: (Left) Rate distortion plot on FastMRI. (Right) COIN++ compression artifacts on FastMRI. See appendix C.12 for more samples.

(see Figure 5.14). As a large number of patches are nearly identical, especially close to the edges, we expect that large gains can be made from modeling the global structure of the data. Qualitatively, our model also performs well although it has patch artifacts at low bitrates (see Figure 5.14).

Applying autoencoder based approaches to this dataset would require non-trivial modifications to the encoder and decoder architectures to model the 3D data. In contrast, for COIN++, we simply change the input dimension of the base network to 3, allowing us to easily model correlations across depth.

5.4 Related Work

Neural Data Compression. Learned compression approaches are typically based on autoencoders that jointly minimize rate and distortion (Ballé et al., 2017). Ballé et al. (2018a) extend this by adding a learned prior while Mentzer et al. (2018), Minnen et al. (2018a), and Lee et al. (2019a) use an autoregressive model to improve entropy coding. Cheng et al. (2020) improve the accuracy of the entropy models by adding attention and Gaussian mixture models for the distribution of latent codes, while Xie et al. (2021) use invertible convolutional layers to enhance performance further.

In a similar vein to work in the latent variable model literature (Hjelm et al., 2016; Krishnan et al., 2018; Kim et al., 2018; Marino et al., 2018), several works (Campos et al., 2019; Guo et al., 2020; Yang et al., 2020) attempt to close the amortization gap (Cremer et al., 2018) by performing iterative gradient-based optimization steps

on top of the use of amortized inference networks. (Yang et al., 2020) additionally identify the discretization gap stemming from the quantization of the latent variables and attempt to close it by inference-time per-instance optimization. (van Rozendaal et al., 2021) take the idea of per-instance optimization of the model further: they perform per-instance finetuning of the decoder and transmit the quantized decoder parameter updates along with the latent code, leading to improved rate-distortion performance.

While most of the methods above are optimized on traditional distortion metrics such as MSE or SSIM, other works have explored the use of generative adversarial networks for optimizing perceptual metrics (Agustsson et al., 2019; Mentzer et al., 2020). Neural compression has also been applied to video (Lu et al., 2019; Goliński et al., 2020; Agustsson et al., 2020) and audio (Kleijn et al., 2018; Valin et al., 2019; Yang et al., 2019; Zeghidour et al., 2021).

Implicit Neural Representations. Representing data with neural networks was originally proposed by Stanley (2007) but has seen a recent surge in interest in the 3D vision community (Park et al., 2019; Niemeyer et al., 2019; Chen et al., 2019). Motivated by the fact that memory requirements of deep voxel representations grow cubically, INRs were proposed to compactly encode high resolution signals (Nguyen-Phuoc et al., 2018; Sitzmann et al., 2019a; Dupont et al., 2020; Mildenhall et al., 2020; Tancik et al., 2020b; Sitzmann et al., 2020b). Implicit representations have been applied to several other computer vision tasks including inverse rendering (Sitzmann et al., 2019b; Mildenhall et al., 2020; Yu et al., 2020), modeling 3D scenes (Atzmon et al., 2020; Jiang et al., 2020; Gropp et al., 2020) and super-resolution (Chen et al., 2020). In addition, distributions of neural function representations have been proposed in the context of generative modeling (Dupont et al., 2021b).

INRs for Compression. In addition to our work several recent works have explored the use of INRs for compression. Davies et al. (2020) encode 3D shapes with neural networks and show that this can reduce memory usage compared with traditional decimated meshes. Chen et al. (2021) represent videos by convolutional neural networks that take as input a time index and output a frame in the video. By pruning, quantizing and entropy coding the weights of this network, the authors achieve compression performance close to standard video codecs. Lee et al. (2021)

meta-learn sparse and parameter efficient initializations for INRs and show that this can reduce the number of parameters required to store an image at a given reconstruction quality, although it is not yet competitive with image codecs such as JPEG. Lu et al. (2021) and Isik et al. (2021) explore the use of INRs for volumetric compression.

Two concurrent works also use function representations for image and video compression (Strümpler et al., 2021; Zhang et al., 2021). Strümpler et al. (2021) meta-learn an MLP initialization and subsequently quantize and entropy code the weights of MLPs fitted to images, leading to large performance gains over COIN. However, their approach still requires tens of thousands of iterations at test time to fully converge while underperforming image codecs like JPEG2000. Zhang et al. (2021) compress frames in videos using INRs (which are quantized and entropy coded) while learning a flow warping to model differences between frames. Results on video benchmarks are promising although the performance still lags behind standard video codecs. To the best of our knowledge, none of these works have considered INRs for building a unified compression framework across data modalities.

5.5 Summary

In this chapter we first proposed a new autoencoder-free method for compressing images by fitting neural networks to pixels and storing the weights of the resulting models. We empirically showed that this simple approach can outperform JPEG at low bit-rates, even without the use of entropy coding. We then built on top of this approach and introduced an improved version using meta-learning that significantly improves the performance in terms of compression and encoding time, while being competitive with well-established codecs such as JPEG. Our approach is the first (to the best of our knowledge) neural codec applicable to multiple modalities.

Chapter 6

Conclusion

In this thesis, we described new methods and insights for using meta-gradients and meta-learning for the compression of models and data in deep learning. We addressed the research questions outlined in Chapter 1 and in the course of doing so several new questions were raised. In this chapter, we discuss some of the limitations of our proposed methods and propose several exciting directions for future directions. We end by summarizing the contributions of this thesis.

6.1 Limitations And Future Directions

Meta-Gradients Computation and Memory Cost. The biggest limitation of methods that use meta-gradients is the extra cost in terms of computation and the required memory. In the case of our proposed pruning method, this is not a major issue as we only do the saliency computation once at the beginning of training. The fact that we typically used 3 to 5 meta steps, allowed us to further do this one-time computation on the CPU where memory significantly more memory is available compared to GPUs. However, in the case of using meta-gradients for quantization robustness, this limitation does make our approach less easy to use and as we discuss in the next point, there is even a good case to be made for computing meta-gradients in a loop for the pruning scenario.

Target-Aware ProsPr. While we have tried to address the trainability gap of pruning formulation using ProsPr, there is another axis that is currently not covered by pruning methods, including ProsPr. Our method could be improved by somehow incorporating the target pruning ratio inside the objective. We are often interested in pruning a specific (and often very large) portion of parameters, but the saliency computation is completely unaware of this number. It would be interesting to explore ways to incorporate this number directly into our objective, for example using a differentiable top-k function (Xie et al., 2020). ProsPr was inspired by approaches in gradient-based meta-learning. Such methods often use an outer loop, in addition to the inner loop that models what we would like to do at deployment time, which is crucial for stabilizing meta-learning. It would be interesting to explore whether a top-k aware function could be used in an outer loop where each iteration has a specific pruning target.

Post-Training ProsPr. We formulated the trainability objective in ProsPr in the context of pruning-at-initialization. However, pruning-at-initialization is still a new area of research and the majority of pruning methods target trained models. In Chapter 2 we discussed how most of these post-training methods ignore gradients and instead focus on computing the second-order terms in the loss expansion. While the gradients of a well-trained model are small, there's no reason to assume that meta-gradients of the loss are small. Given that almost all post-training pruning methods also follow pruning with a training stage (fine-tuning) the arguments we made in favor of the goal of ProsPr also apply to post-training pruning. A natural extension of ProsPr would be to evaluate its feasibility for post-training pruning.

Formal Verification for Compression Robustness. A radically different approach toward quantization robustness that goes beyond first-order robustness and does not suffer the implementation challenges of our approach is adopting formal verification for quantization robustness. We briefly dabbled with this idea but leave thorough exploration to future work.

Verification refers to the process of ensuring that the output of a neural network satisfies a certain desirable property. Here we focus on the case where the desired

output property (and the constraints that have to be met) are *linear*:

$$c^\top x_L + d \leq 0 \tag{6.1}$$

Many interesting properties can be formulated as above e.g. robustness to adversarial examples, having monotonic predictors, and cardinality constraints. We do care about adversarial robustness as it pursues the same objective that we care about in quantization robustness, i.e. making sure the prediction is not changed. Changing the output label from the true label i to label j as a result of perturbation of the input can be written as:

$$c^\top x_L \leq 0, \tag{6.2}$$

where $c_j = 1$ and $c_i = -1$ and all other components are zero. We can often write this as an optimization problem. For instance, if we want to verify the behavior of the network over some set of the input S then we can write this as

$$\begin{aligned} \text{minimise } \; c^\top x_L \quad & w.r.t \;\; \{x_i, z_i\} \\ & \text{subject to.} \\ x_0 &= S(x_{in}) \\ z_{i+1} &= W_i x_i + b_i \\ x_i &= \max(z_i, 0) \quad \forall i \in 1..L, \end{aligned}$$

where we assume ReLU activations and x_i is the input to the i^{th} layer, z_i is the pre-activation of the i^{th} layer, and W and b are the weights and biases of that layer with L layers in total. The minimization of the linear function $c^\top x$ can be seen as verifying an assertion of the form $c^\top x + d \geq 0$.

In general this is a non-convex optimization and NP-hard and scaling it up for neural networks is not trivial but Dvijotham et al. (2018a,b) have recently managed to apply this approach to relatively large neural networks for adversarial robustness. One way to make the problem more tractable is using Lagrangian relaxation where we allow constraints to be violated and penalize violations using Lagrange multipliers, which impose a cost on violations. These added costs are used instead of the strict inequality constraints in the optimization.

So the original problem

$$\begin{aligned} & c^T x \\ \text{s.t.} \quad & Ax \leq b, \end{aligned} \tag{6.3}$$

turns into:

$$\max \quad c^T x + \lambda^T (b - Ax). \tag{6.4}$$

The application of this approach to quantization, and even pruning, is clear - if we have managed to train a verified network of this kind, we are able to quantize it or prune to our heart's content, provided we never move the weights more than ϵ from $\hat{W}$ in the ℓ_∞ norm (note that this includes pruning because if $\hat{W}$ is within ϵ of 0 we are totally within our rights to prune it entirely rather than quantize). In the simple case of a single neural network, the problem can be formulated as

$$\begin{aligned} \text{minimise} \;\; c^\top x_o \quad \text{w.r.t} \;\; & \{x_o, z_o, W, b\} \\ \text{subject to.} & \\ z_o &= W x_{in} + b \\ x_i &= \max(z_0, 0) \\ \hat{W} - \epsilon \leq W &\leq \hat{W} + \epsilon \\ \hat{b} - \epsilon \leq b &\leq \hat{b} + \epsilon \end{aligned}$$

The main challenge is that unlike input perturbation where the constraint $z_i = W_i x_i + b_i$ in their setting is linear, as W is fixed, in our case after the first layer this involved two variables W and x being coupled together quadratically. This makes the problem in the dual considerably more difficult to solve. In particular, the Lagrangian requires us to solve a maximization over all the primal variables.

Data Compression Future Work. In its current form our method employs very basic methods for both quantization and entropy coding - using more sophisticated techniques for these two steps could likely lead to large performance gains. Recent success in modeling distributions of functions (Schwarz et al., 2020; Anokhin et al., 2021; Skorokhodov et al., 2021; Dupont et al., 2021b) suggests that large gains could be made from using deep generative models to learn the distribution of modulations for entropy coding. Similarly, better post-training quantization (Nagel et al., 2019a; Li et al., 2021) or quantization-aware training (Krishnamoorthi, 2018b; Esser et al.,

2020) are also likely to improve performance. For large scale image data, it would be interesting to model the global structure of patches instead of encoding and entropy coding them independently. Further, the field of INRs is progressing rapidly and these advances are likely to improve COIN++ too. For example, Martel et al. (2021) use adaptive patches to scale INRs to gigapixel images - such a partition of the input is similar to the variable size blocks used in BPG (Bellard, 2014). In addition, using better activation functions (Ramasinghe et al., 2021) to increase PSNR and equilibrium models (Huang et al., 2021) to reduce memory usage are exciting avenues for future research.

6.2 Summary

This thesis addressed the question of using meta-gradient for Deep Learning compression. We first looked at compressing Deep Learning models and motivated the use of information in meta-gradients to improve pruning and quantization of neural networks in various vision classification tasks. In Chapter 3 we identified a blind spot in formulation of saliency-based pruning methods and argued that their saliency objectives ignore the fact that pruning is often followed by more training. We then showed how meta-gradients can be used to explicitly capture what we called trainability of parameters and developed a new method for pruning-at-initialization that obtains excellent performance. A difficult open question is how to reduce the computation complexity of meta-gradients to capture longer-term training dynamics in the pruning objective and how to reduce the noise in this single-shot signal. In Chapter 4 we looked at quantizing Deep Learning models and instead of targeting a particular bit-width we provided a model for quantization noise and showed how meta-gradients can be used as a regularization term to instead learn models that are inherently more robust against naive post-training quantization. In Chapter 5 we switched to data compression using Deep Learning and laid the foundation for autoencoder-free data compression using implicit neural representations. We then proposed an improved version of this method and show how meta-gradients can be used to learn a base network for quick data compression. We show that this method is applicable to a wide range of data modalities, from images and audio to medical and climate data.

References

Aaron, Anne, Zhi Li, Megha Manohara, Jan De Cock, and David Ronca (2015). *Per-Title Encode Optimization*. [Online; accessed 26-Feb-2021] (cit. on p. 71).

Achterhold, Jan, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein (2018). "Variational network quantization". In: *International Conference on Learning Representations* (cit. on p. 67).

Agustsson, Eirikur, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici (2020). "Scale-Space Flow for End-to-End Optimized Video Compression". In: *CVPR* (cit. on p. 87).

Agustsson, Eirikur, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool (2019). "Generative adversarial networks for extreme learned image compression". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 221–231 (cit. on p. 87).

Anokhin, Ivan, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov (2021). "Image generators with conditionally-independent pixel synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14278–14287 (cit. on p. 92).

Antoniou, Antreas, Harrison Edwards, and Amos Storkey (2019). *How to train your MAML*. arXiv: 1810.09502 [cs.LG] (cit. on pp. 21, 48, 51).

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (cit. on p. 22).

Atzmon, Matan and Yaron Lipman (2020). "Sal: Sign agnostic learning of shapes from raw data". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2565–2574 (cit. on p. 87).

Ballé, Johannes, Valero Laparra, and Eero P Simoncelli (2017). "End-to-end optimized image compression". In: *International Conference on Learning Representations* (cit. on p. 86).

Ballé, Johannes, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston (2018a). "Variational Image Compression with a Scale Hyperprior". In: *International Conference on Learning Representations* (cit. on pp. 69, 78, 81, 86).

— (2018b). "Variational image compression with a scale hyperprior". In: *International Conference on Learning Representations* (cit. on p. 5).

Banner, R, Y Nahshan, E Hoffer, and D Soudry (2018). "Post training 4-bit quantization of convolution networks for rapid-deployment". In: *CoRR, abs/1810.05723* 1, p. 2 (cit. on p. 54).

Basri, Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman (2019). "The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies". In: *Advances in Neural Information Processing Systems* (cit. on p. 70).

Baydin, Atilim Gunes, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18.153 (cit. on p. 61).

Bégaint, Jean, Fabien Racapé, Simon Feltman, and Akshay Pushparaja (2020). "CompressAI: a PyTorch library and evaluation platform for end-to-end compression research". In: *arXiv preprint arXiv:2011.03029* (cit. on pp. 78, 133, 136).

Bellard, Fabrice (2014). *BPG image format*. https://bellard.org/bpg/. [Online; accessed 27-Dec-2021] (cit. on pp. 81, 93, 137).

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville (2013a). *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation* (cit. on p. 4).

— (2013b). "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432* (cit. on pp. 20, 54).

Bhattiprolu, Vijay, Mrinalkanti Ghosh, Venkatesan Guruswami, Euiwoong Lee, and Madhur Tulsiani (2018). "Inapproximability of Matrix $p \to q$ Norms". In: *arXiv preprint arXiv:1802.07425* (cit. on p. 129).

Bross, Benjamin, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm (2021). "Overview of the versatile video coding (VVC) standard and its applications". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.10, pp. 3736–3764 (cit. on p. 81).

Cai, Zhaowei and Nuno Vasconcelos (June 2020). "Rethinking Differentiable Search for Mixed-Precision Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 20).

Campos, Joaquim, Simon Meierhans, Abdelaziz Djelouah, and Christopher Schroers (2019). "Content Adaptive Optimization for Neural Image Compression". In: *CVPR Workshop on Learned Image Compression* (cit. on p. 86).

Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello (2016). "An analysis of deep neural network models for practical applications". In: *arXiv preprint arXiv:1605.07678* (cit. on p. 2).

Chan, Eric R, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein (2021). "pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5799–5809 (cit. on pp. 73, 74).

Chen, Hao, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava (2021). "Nerv: Neural representations for videos". In: *Advances in Neural Information Processing Systems* (cit. on p. 87).

Chen, Yinbo, Sifei Liu, and Xiaolong Wang (2020). "Learning Continuous Image Representation with Local Implicit Image Function". In: *arXiv preprint arXiv:2012.09161* (cit. on p. 87).

Chen, Zhiqin and Hao Zhang (2019). "Learning Implicit Fields for Generative Shape Modeling". In: *CVPR* (cit. on p. 87).

Cheng, Zhengxue, Heming Sun, Masaru Takeuchi, and Jiro Katto (2020). "Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules". In: *CVPR* (cit. on pp. 78, 81, 86).

Choukroun, Yoni, Eli Kravchik, and Pavel Kisilev (2019). "Low-bit quantization of neural networks for efficient inference". In: *arXiv preprint arXiv:1902.06822* (cit. on p. 54).

Cisse, Moustapha, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier (2017). "Parseval networks: Improving robustness to adversarial examples". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 854–863 (cit. on p. 66).

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David (2015). "Binaryconnect: Training deep neural networks with binary weights during propagations". In: *Advances in neural information processing systems*, pp. 3123–3131 (cit. on pp. 16, 67).

Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio (2016). "Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-1". In: *arXiv preprint arXiv:1602.02830* (cit. on p. 16).

Cremer, Chris, Xuechen Li, and David Duvenaud (2018). "Inference Suboptimality in Variational Autoencoders". In: *ICML* (cit. on p. 86).

Davies, Thomas, Derek Nowrouzezahrai, and Alec Jacobson (2020). "On the effectiveness of weight-encoded neural implicit 3D shapes". In: *arXiv preprint arXiv:2009.09808* (cit. on p. 87).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186 (cit. on p. 15).

Dong, Zhen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer (2019). "HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision". In: *arXiv preprint arXiv:1905.03696* (cit. on p. 20).

Dupont, Emilien, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet (2021a). "COIN: COmpression with Implicit Neural representations". In: *arXiv preprint arXiv:2103.03123* (cit. on pp. 72, 77, 81, 136).

Dupont, Emilien, Miguel Bautista Martin, Alex Colburn, Aditya Sankar, Josh Susskind, and Qi Shan (2020). "Equivariant neural rendering". In: *International Conference on Machine Learning*. PMLR, pp. 2761–2770 (cit. on p. 87).

Dupont, Emilien, Yee Whye Teh, and Arnaud Doucet (2021b). "Generative Models as Distributions of Functions". In: *arxiv preprint arXiv:2102.04776* (cit. on pp. 83, 87, 92, 134).

Dvijotham, Krishnamurthy, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli (2018a). "Training verified learners with learned verifiers". In: *arXiv preprint arXiv:1805.10265* (cit. on p. 91).

Dvijotham, Krishnamurthy, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli (2018b). "A Dual Approach to Scalable Verification of Deep Networks." In: *UAI*. Vol. 1. 2, p. 3 (cit. on p. 91).

Esser, Steven K., Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha (2020). "Learned Step Size Quantization". In: *International Conference on Learning Representations* (cit. on p. 92).

Evans, C, I Julian, and F Simon (2020). "The Sustainable Future of Video Entertainment from Creation to Consumption". In: *Futuresource Consulting Ltd.: Hertfordshire, UK*, pp. 1–34 (cit. on p. 23).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (Aug. 2017a). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1126–1135 (cit. on p. 21).

— (2017b). "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International Conference on Machine Learning* (cit. on pp. 75, 77, 145).

Frankle, Jonathan and Michael Carbin (2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations* (cit. on pp. 14, 25, 52).

Frankle, Jonathan, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin (2021). "Pruning Neural Networks at Initialization: Why Are We Missing the Mark?" In: *International Conference on Learning Representations* (cit. on pp. 34, 46, 47, 119).

Frankle, Jonathan, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin (2020). *Linear Mode Connectivity and the Lottery Ticket Hypothesis*. arXiv: 1912.05671 [cs.LG] (cit. on pp. 26, 46).

Gal, Yarin, Riashat Islam, and Zoubin Ghahramani (2017). "Deep Bayesian Active Learning with Image Data". In: *Proceedings of The 34th International Conference on Machine Learning* (cit. on p. 118).

Gale, Trevor, Erich Elsen, and Sara Hooker (2019). "The state of sparsity in deep neural networks". In: *arXiv preprint arXiv:1902.09574* (cit. on pp. 12, 52).

Gilpin, Leilani H., David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal (2019). *Explaining Explanations: An Overview of Interpretability of Machine Learning*. arXiv: 1806.00069 [cs.AI] (cit. on p. 11).

Goliński, Adam, Reza Pourreza, Yang Yang, Guillaume Sautiére, and Taco S. Cohen (2020). "Feedback Recurrent Autoencoder for Video Compression". In: *Proceedings of the Asian Conference on Computer Vision* (cit. on p. 87).

Gomez, Aidan N, Ivan Zhang, Siddhartha Rao Kamalakara, Divyam Madaan, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton (2019). "Learning sparse networks using targeted dropout". In: *arXiv preprint arXiv:1905.13678* (cit. on p. 13).

Gordon, Ariel, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi (June 2018). "MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks". en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, pp. 1586–1595 (cit. on p. 52).

Grill, Jean-Bastien, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu koray, Remi Munos, and Michal Valko (2020). "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 21271–21284 (cit. on p. 123).

Gropp, Amos, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman (2020). "Implicit geometric regularization for learning shapes". In: *arXiv preprint arXiv:2002.10099* (cit. on p. 87).

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville (2017). "Improved training of wasserstein gans". In: *Advances in neural information processing systems*, pp. 5767–5777 (cit. on pp. 22, 60, 64, 66).

Guo, Tiansheng, Jing Wang, Ze Cui, Yihui Feng, Yunying Ge, and Bo Bai (2020). "Variable Rate Image Compression with Content Adaptive Optimization". In: *CVPR Workshops* (cit. on p. 86).

Guo, Yiwen, Chao Zhang, Changshui Zhang, and Yurong Chen (2018). "Sparse dnns with improved adversarial robustness". In: *Advances in neural information processing systems*, pp. 242–251 (cit. on p. 67).

Gupta, Suyog, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan (2015). "Deep learning with limited numerical precision". In: *International Conference on Machine Learning*, pp. 1737–1746 (cit. on p. 67).

Gysel, Philipp (2016). "Ristretto: Hardware-oriented approximation of convolutional neural networks". In: *arXiv preprint arXiv:1605.06402* (cit. on p. 67).

Hamblin, Chris, Talia Konkle, and George Alvarez (2022). *Pruning for Interpretable, Feature-Preserving Circuits in CNNs* (cit. on p. 11).

Hassibi, Babak and David G Stork (1993a). "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in neural information processing systems*, pp. 164–171 (cit. on p. 26).

Hassibi, Babak, David G Stork, and Gregory J Wolff (1993b). "Optimal brain surgeon and general network pruning". In: *IEEE international conference on neural networks*. IEEE, pp. 293–299 (cit. on pp. 2, 10).

Hayou, Soufiane, Jean-Francois Ton, Arnaud Doucet, and Yee Whye Teh (2021). "Robust Pruning at Initialization". In: *International Conference on Learning Representations* (cit. on p. 51).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034 (cit. on pp. 31, 32, 116).

— (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on pp. 28, 30, 34).

— (2016b). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on p. 60).

He, Yihui, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han (Jan. 2019). "AMC: AutoML for Model Compression and Acceleration on Mobile Devices". In: *Computer Vision and Pattern Recognition*. arXiv: 1802.03494 (cit. on p. 52).

He, Yihui, Xiangyu Zhang, and Jian Sun (Oct. 2017). "Channel Pruning for Accelerating Very Deep Neural Networks". en. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, pp. 1398–1406 (cit. on p. 52).

Hendrickx, Julien M. and Alex Olshevsky (2010). "Matrix p-Norms Are NP-Hard to Approximate If $p \neq 1, 2, \infty$". In: *SIAM Journal on Matrix Analysis and Applications* 31.5, pp. 2802–2812 (cit. on pp. 59, 129).

Hersbach, H., B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee, and J-N. Thépaut (2019). *ERA5 monthly averaged data on single levels from 1979 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS).* https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-monthly-means (Accessed 27-09-2021) (cit. on pp. 83, 134).

Hinton, Geoffrey E and Drew Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights". In: *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13 (cit. on p. 10).

Hjelm, R. Devon, Kyunghyun Cho, Junyoung Chung, Russ Salakhutdinov, Vince Calhoun, and Nebojsa Jojic (2016). "Iterative Refinement of the Approximate Posterior for Directed Belief Networks". In: *NeurIPS* (cit. on p. 86).

Hoeffding, Wassily (Mar. 1963). "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301, pp. 13–30 (cit. on p. 127).

Hoffman, Judy, Daniel A. Roberts, and Sho Yaida (Aug. 2019). "Robust Learning with Jacobian Regularization". In: *arXiv:1908.02729 [cs, stat]*. arXiv: 1908.02729 (cit. on pp. 58, 62).

Huang, Zhichun, Shaojie Bai, and J Zico Kolter (2021). "Implicit^2: Implicit Layers for Implicit Representations". In: *Advances in Neural Information Processing Systems* (cit. on p. 93).

Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio (2017). "Quantized neural networks: Training neural networks with low precision

weights and activations". In: *The Journal of Machine Learning Research* 18.1, pp. 6869–6898 (cit. on p. 16).

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. JMLR. org, pp. 448–456 (cit. on p. 37).

Isik, Berivan, Philip A Chou, Sung Jin Hwang, Nick Johnston, and George Toderici (2021). "LVAC: Learned Volumetric Attribute Compression for Point Clouds using Coordinate Based Networks". In: *arXiv preprint arXiv:2111.08988* (cit. on p. 88).

Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko (June 2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 21, 61, 67).

Jeong, Taehee, Manasa Bollavaram, Elliott Delaye, and Ashish Sirasao (2021). "Neural network pruning for biomedical image segmentation". In: *Medical Imaging 2021: Image-Guided Procedures, Robotic Interventions, and Modeling*. Vol. 11598. International Society for Optics and Photonics, 115981J (cit. on pp. 122, 123).

Jiang, Chiyu, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. (2020). "Local implicit grid representations for 3d scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6001–6010 (cit. on p. 87).

Jordao, Artur and Hélio Pedrini (2021). "On the Effect of Pruning on Adversarial Robustness". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1–11 (cit. on p. 11).

Jorge, Pau de, Amartya Sanyal, Harkirat Behl, Philip Torr, Grégory Rogez, and Puneet K. Dokania (2021). "Progressive Skeletonization: Trimming more fat from a network at initialization". In: *International Conference on Learning Representations* (cit. on pp. 26, 35, 48, 49, 51, 119).

Kandasamy, Kirthevasan, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing (2020).

"Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly." In: *J. Mach. Learn. Res.* 21.81, pp. 1–27 (cit. on p. 23).

Kaya, Yigitcan, Sanghyun Hong, and Tudor Dumitras (2018). *Shallow-Deep Networks: Understanding and Mitigating Network Overthinking* (cit. on p. 11).

Kim, Yoon, Sam Wiseman, Andrew C. Miller, David Sontag, and Alexander M. Rush (2018). "Semi-Amortized Variational Autoencoders". In: *ICML* (cit. on p. 86).

Kleijn, W Bastiaan, Felicia SC Lim, Alejandro Luebs, Jan Skoglund, Florian Stimberg, Quan Wang, and Thomas C Walters (2018). "Wavenet based low rate speech coding". In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 676–680 (cit. on p. 87).

Knoll, Florian, Jure Zbontar, Anuroop Sriram, Matthew J Muckley, Mary Bruno, Aaron Defazio, Marc Parente, Krzysztof J Geras, Joe Katsnelson, Hersh Chandarana, et al. (2020). "fastMRI: A publicly available raw k-space and DICOM dataset of knee images for accelerated MR image reconstruction using machine learning". In: *Radiology: Artificial Intelligence* 2.1, e190007 (cit. on p. 134).

Kodak (1991). *Kodak Dataset*. http://r0k.us/graphics/kodak/ (cit. on pp. 72, 78, 84).

Krishnamoorthi, Raghuraman (2018a). "Quantizing deep convolutional networks for efficient inference: A whitepaper". In: *arXiv preprint arXiv:1806.08342* (cit. on pp. 21, 54, 61).

— (2018b). "Quantizing deep convolutional networks for efficient inference: A whitepaper". In: *arxiv preprint arxiv:1806.08342* (cit. on p. 92).

Krishnan, Rahul G., Dawen Liang, and Matthew Hoffman (2018). "On the Challenges of Learning with Inference Networks on Sparse, High-Dimensional Data". In: *AISTATS* (cit. on p. 86).

Kuhn, Lorenz, Clare Lyle, Aidan N. Gomez, Jonas Rothfuss, and Yarin Gal (2021). *Robustness to Pruning Predicts Generalization in Deep Neural Networks*. arXiv: 2103.06002 [cs.LG] (cit. on p. 11).

Laurent, César, Camille Ballas, Thomas George, Nicolas Ballas, and Pascal Vincent (2020). *Revisiting Loss Modelling for Unstructured Pruning* (cit. on p. 15).

LeCun, Yann, John S Denker, and Sara A Solla (1990a). "Optimal brain damage". In: *Advances in neural information processing systems*, pp. 598–605 (cit. on pp. 2, 10, 13).

LeCun, Yann, John S Denker, and Sara A Solla (1990b). "Optimal brain damage". In: *Advances in neural information processing systems*, pp. 598–605 (cit. on p. 26).

Lee, Jaeho, Jihoon Tack, Namhoon Lee, and Jinwoo Shin (2021). "Meta-Learning Sparse Implicit Neural Representations". In: *Advances in Neural Information Processing Systems* 34 (cit. on p. 87).

Lee, Jooyoung, Seunghyun Cho, and Seung-Kwon Beack (2019a). "Context-Adaptive Entropy Model for End-to-End Optimized Image Compression". In: *International Conference on Learning Representations* (cit. on pp. 69, 86).

— (2019b). "Context-adaptive Entropy Model for End-to-end Optimized Image Compression". In: *International Conference on Learning Representations* (cit. on p. 5).

Lee, Namhoon, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr (2019c). "A Signal Propagation Perspective for Pruning Neural Networks at Initialization". In: *International Conference on Learning Representations* (cit. on p. 51).

Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip Torr (2019d). "SNIP: Single-Shot Network Pruning Based on Connection Sensitivity". In: *International Conference on Learning Representations* (cit. on pp. 26, 27, 37, 40, 51).

Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf (Nov. 2016). "Pruning Filters for Efficient ConvNets". In: *International Conference on Learning Representations* (cit. on p. 52).

Li, Yuhang, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu (2021). "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction". In: *International Conference on Learning Representations* (cit. on p. 92).

Lin, Ji, Chuang Gan, and Song Han (2019). "Defensive quantization: When efficiency meets robustness". In: *arXiv preprint arXiv:1904.08444* (cit. on pp. 60–63, 65, 66, 130).

Liu, Zechun, Kwang-Ting Cheng, Dong Huang, Eric Xing, and Zhiqiang Shen (2022). "Nonuniform-to-Uniform Quantization: Towards Accurate Quantization via Generalized Straight-Through Estimation". In: (cit. on p. 19).

Liu, Zechun, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng (2020). "Bi-real net: Binarizing deep network towards real-network performance". In: *International Journal of Computer Vision* 128.1, pp. 202–219 (cit. on p. 16).

Liu, Zhuang, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang (2017). "Learning efficient convolutional networks through network slimming". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744 (cit. on p. 52).

Liu, Zhuang, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell (2019). "Rethinking the Value of Network Pruning". In: *International Conference on Learning Representations* (cit. on pp. 41, 52).

Lorraine, Jonathan, Paul Vicol, and David Duvenaud (2020). "Optimizing millions of hyperparameters by implicit differentiation". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1540–1552 (cit. on p. 23).

Louizos, Christos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling (2018). "Relaxed quantization for discretized neural networks". In: *arXiv preprint arXiv:1810.01875* (cit. on pp. 54, 67).

Louizos, Christos, Max Welling, and Diederik P Kingma (2017). "Learning Sparse Neural Networks through L_0 Regularization". In: *arXiv preprint arXiv:1712.01312* (cit. on p. 13).

Lu, Guo, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao (2019). "Dvc: An end-to-end deep video compression framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11006–11015 (cit. on p. 87).

Lu, Yuzhe, Kairong Jiang, Joshua A Levine, and Matthew Berger (2021). "Compressive neural representations of volumetric scalar fields". In: *Computer Graphics Forum*. Vol. 40. 3. Wiley Online Library, pp. 135–146 (cit. on p. 88).

Luo, Jian-Hao, Jianxin Wu, and Weiyao Lin (2017). "Thinet: A filter level pruning method for deep neural network compression". In: *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066 (cit. on p. 52).

Lym, Sangkug, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez (Nov. 2019). "PruneTrain". In: *Proceedings of the International*

Conference for High Performance Computing, Networking, Storage and Analysis (cit. on p. 52).

Maclaurin, Dougal, David Duvenaud, and Ryan Adams (2015). "Gradient-based hyperparameter optimization through reversible learning". In: *International conference on machine learning*. PMLR, pp. 2113–2122 (cit. on p. 23).

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh (2017). "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *International Conference on Learning Representations* (cit. on p. 13).

Manning, Christopher D, Prabhakar Raghavan, and Hinrich Schütze (2009). *Introduction to Information Retrieval*. en. Cambridge University Press (cit. on p. 33).

Marino, Joseph, Yisong Yue, and Stephan Mandt (2018). "Iterative Amortized Inference". In: *ICML* (cit. on p. 86).

Martel, Julien NP, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein (2021). "ACORN: Adaptive Coordinate Networks for Neural Scene Representation". In: *arXiv preprint arXiv:2105.02788* (cit. on p. 93).

Mehta, Ishit, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker (2021). "Modulated Periodic Activations for Generalizable Local Functional Representations". In: *arXiv preprint arXiv:2104.03960* (cit. on pp. 73, 74, 145).

Meller, Eldad, Alexander Finkelstein, Uri Almog, and Mark Grobman (2019). "Same, Same But Different - Recovering Neural Network Quantization Error Through Weight Factorization". In: *International Conference on Machine Learning* (cit. on p. 54).

Mentzer, Fabian, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool (2018). "Conditional probability models for deep image compression". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402 (cit. on p. 86).

Mentzer, Fabian, George Toderici, Michael Tschannen, and Eirikur Agustsson (2020). "High-fidelity generative image compression". In: *arXiv preprint arXiv:2006.09965* (cit. on p. 87).

Micikevicius, Paulius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev,

Ganesh Venkatesh, et al. (2017). “Mixed precision training”. In: *arXiv preprint arXiv:1710.03740* (cit. on p. 20).

Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng (2020). “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *ECCV* (cit. on pp. 70, 71, 87).

Minnen, David, Johannes Ballé, and George Toderici (2018a). “Joint Autoregressive and Hierarchical Priors for Learned Image Compression”. In: *Advances in Neural Information Processing Systems* (cit. on pp. 69, 78, 81, 86).

Minnen, David, Johannes Ballé, and George D Toderici (2018b). “Joint Autoregressive and Hierarchical Priors for Learned Image Compression”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc. (cit. on p. 5).

Močkus, Jonas (1975). “On Bayesian methods for seeking the extremum”. In: *Optimization techniques IFIP technical conference*. Springer, pp. 400–404 (cit. on p. 23).

Molchanov, Dmitry, Arsenii Ashukha, and Dmitry Vetrov (2017). “Variational dropout sparsifies deep neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2498–2507 (cit. on p. 13).

Mozer, Michael C and Paul Smolensky (1988). “Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 1. Morgan-Kaufmann (cit. on p. 10).

MP3 (1993). *MP3 codec*. https://www.iso.org/standard/22412.html (cit. on pp. 81, 84).

Nagel, Markus, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort (2020). *Up or Down? Adaptive Rounding for Post-Training Quantization* (cit. on p. 17).

Nagel, Markus, Mart van Baalen, Tijmen Blankevoort, and Max Welling (2019a). “Data-Free Quantization Through Weight Equalization and Bias Correction”. In: *arXiv preprint arXiv:1906.04721* (cit. on p. 92).

Nagel, Markus, Mart van Baalen, Tijmen Blankevoort, and Max Welling (2019b). "Data-Free Quantization through Weight Equalization and Bias Correction". In: *arXiv preprint arXiv:1906.04721* (cit. on pp. 19, 54, 62).

Nagel, Markus, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort (2021). "A white paper on neural network quantization". In: *arXiv preprint arXiv:2106.08295* (cit. on pp. 16, 19).

Nguyen-Phuoc, Thu, Chuan Li, Stephen Balaban, and Yong-Liang Yang (2018). "RenderNet: A Deep Convolutional Network for Differentiable Rendering from 3D Shapes". In: *NeurIPS* (cit. on p. 87).

Nichol, Alex, Joshua Achiam, and John Schulman (2018). "On first-order meta-learning algorithms". In: *arXiv preprint arXiv:1803.02999* (cit. on pp. 77, 145).

Niemeyer, Michael, Lars Mescheder, Michael Oechsle, and Andreas Geiger (2019). "Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics". In: *ICCV* (cit. on p. 87).

Panayotov, Vassil, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur (2015). "Librispeech: an asr corpus based on public domain audio books". In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, pp. 5206–5210 (cit. on pp. 84, 135).

Park, Jeong Joon, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove (2019). "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CVPR* (cit. on p. 87).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019a). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035 (cit. on pp. 37, 38).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison,

Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019b). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* (cit. on pp. 78, 80).

Perez, Ethan, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville (2018). "Film: Visual reasoning with a general conditioning layer". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1 (cit. on p. 73).

Peters, Jorn WT and Max Welling (2018). "Probabilistic binary neural networks". In: *arXiv preprint arXiv:1809.03368* (cit. on p. 19).

Rajeswaran, Aravind, Chelsea Finn, Sham Kakade, and Sergey Levine (2019). "Meta-learning with implicit gradients". In: *Advances in Neural Information Processing Systems* (cit. on p. 77).

Ramasinghe, Sameera and Simon Lucey (2021). "Beyond Periodicity: Towards a Unifying Framework for Activations in Coordinate-MLPs". In: *arXiv preprint arXiv:2111.15135* (cit. on p. 93).

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi (2016). "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer, pp. 525–542 (cit. on pp. 16, 67).

Reagen, Brandon, Robert Adolf, Paul Whatmough, Gu-Yeon Wei, and David Brooks (2017). "Deep Learning for Computer Architects". In: *Synthesis Lectures on Computer Architecture* 12.4, pp. 1–123 (cit. on p. 3).

Renda, Alex, Jonathan Frankle, and Michael Carbin (2020). *Comparing Rewinding and Fine-tuning in Neural Network Pruning*. arXiv: 2003.02389 [cs.LG] (cit. on p. 46).

Rissanen, Jorma and Glen G Langdon (1979). "Arithmetic coding". In: *IBM Journal of Research and Development* 23.2, pp. 149–162 (cit. on p. 77).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241 (cit. on p. 122).

Schoenholz, Samuel S, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein (2016). "Deep Information Propagation". In: *International Conference on Learning Representations* (cit. on p. 51).

Schwarz, Katja, Yiyi Liao, Michael Niemeyer, and Andreas Geiger (2020). "Graf: Generative radiance fields for 3d-aware image synthesis". In: *arXiv preprint arXiv:2007.02442* (cit. on p. 92).

Sedghi, Hanie, Vineet Gupta, and Philip M Long (2018). "The singular values of convolutional layers". In: *arXiv preprint arXiv:1805.10408* (cit. on p. 67).

Settles, Burr (2010). "Active Learning Literature Survey". In: *Machine Learning* 15.2, pp. 201–221. arXiv: 1206.5533 (cit. on pp. 33, 44, 117).

Shayer, Oran, Dan Levi, and Ethan Fetaya (2017). "Learning discrete weights using the local reparameterization trick". In: *arXiv preprint arXiv:1710.07739* (cit. on p. 67).

Sicuranza, Giovanni L., G. Romponi, and Stefano Marsi (1990). "Artificial neural network for image compression". In: *Electronics Letters* 26, pp. 477–479 (cit. on p. 5).

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (cit. on p. 60).

— (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations* (cit. on pp. 28, 42).

Singh, Sidak Pal and Dan Alistarh (2020). "WoodFisher: Efficient Second-Order Approximation for Neural Network Compression". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 18098–18109 (cit. on p. 13).

Sitzmann, Vincent, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein (2020a). "MetaSDF: Meta-Learning Signed Distance Functions". In: *Advances in Neural Information Processing Systems* (cit. on pp. 72, 75).

Sitzmann, Vincent, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein (2020b). "Implicit Neural Representations with Periodic Activation Functions". In: *Advances in Neural Information Processing Systems* (cit. on pp. 24, 70, 71, 87, 135, 139).

Sitzmann, Vincent, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer (2019a). "DeepVoxels: Learning Persistent 3D Feature Embeddings". In: *CVPR* (cit. on p. 87).

Sitzmann, Vincent, Michael Zollhöfer, and Gordon Wetzstein (2019b). "Scene representation networks: Continuous 3d-structure-aware neural scene representations". In: *Advances in Neural Information Processing Systems* (cit. on p. 87).

Skodras, Athanassios, Charilaos Christopoulos, and Touradj Ebrahimi (2001). "The jpeg 2000 still image compression standard". In: *IEEE Signal Processing Magazine* 18.5, pp. 36–58 (cit. on pp. 81, 137).

Skorokhodov, Ivan, Savva Ignatyev, and Mohamed Elhoseiny (2021). "Adversarial generation of continuous images". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10753–10764 (cit. on p. 92).

Sonehara, Kawato, Miyake, and Nakane (1989). "Image data compression using a neural network model". In: *International 1989 Joint Conference on Neural Networks*, 35–41 vol.2 (cit. on p. 5).

Sonehara, N (1989). "Image data compression using a neural network model". In: *Proc. Int. Joint Conf. on Neural Networks*, pp. II–35 (cit. on p. 24).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958 (cit. on p. 13).

Stanley, Kenneth O (2007). "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic Programming and Evolvable Machines* 8.2, pp. 131–162 (cit. on p. 87).

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (2019). "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650 (cit. on pp. 15, 52).

Strümpler, Yannick, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari (2021). "Implicit Neural Representations for Image Compression". In: *arXiv preprint arXiv:2112.04267* (cit. on p. 88).

Sze, V., Y.H. Chen, T.J. Yang, and J.S. Emer (2020). *Efficient Processing of Deep Neural Networks*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers (cit. on p. 3).

Sze, Vivienne, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer (2017). "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12, pp. 2295–2329 (cit. on p. 14).

Tanaka, Hidenori, Daniel Kunin, Daniel L Yamins, and Surya Ganguli (2020). "Pruning neural networks without any data by iteratively conserving synaptic flow". In: *Advances in Neural Information Processing Systems* 33 (cit. on pp. 26, 47, 51).

Tancik, Matthew, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng (2020a). "Learned Initializations for Optimizing Coordinate-Based Neural Representations". In: *arXiv:2012.02189 [cs]* (cit. on pp. 72, 75).

Tancik, Matthew, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng (2020b). "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains". In: *NeurIPS* (cit. on pp. 70, 71, 87).

Theis, Lucas, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár (2018). "Faster gaze prediction with dense networks and fisher pruning". In: *arXiv preprint arXiv:1801.05787* (cit. on pp. 13, 52).

Tran, Brandon, Jerry Li, and Aleksander Madry (2018). "Spectral Signatures in Backdoor Attacks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc. (cit. on p. 11).

Valin, Jean-Marc and Jan Skoglund (2019). "LPCNet: Improving neural speech synthesis through linear prediction". In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 5891–5895 (cit. on p. 87).

van Rozendaal, Ties, Iris A M Huijben, and Taco S Cohen (2021). "Overfitting for Fun and Profit: Instance-Adaptive Data Compression". In: *ICLR* (cit. on p. 87).

Veniat, Tom and Ludovic Denoyer (May 2018). "Learning Time/Memory-Efficient Deep Architectures with Budgeted Super Networks". en. In: *Computer Vision and Pattern Recognition*. arXiv: 1706.00046 (cit. on p. 52).

Wallace, Gregory K (1992). "The JPEG still picture compression standard". In: *IEEE Transactions on Consumer Electronics* 38.1, pp. xviii–xxxiv (cit. on pp. 81, 137).

Wang, Chaoqi, Roger Grosse, Sanja Fidler, and Guodong Zhang (2019). "EigenDamage: Structured Pruning in the Kronecker-Factored Eigenbasis". In: *International Conference on Machine Learning*, pp. 6566–6575 (cit. on pp. 11, 13, 31, 53).

Wang, Chaoqi, Guodong Zhang, and Roger Grosse (2020a). "Picking Winning Tickets Before Training by Preserving Gradient Flow". In: *International Conference on Learning Representations* (cit. on pp. 26, 27, 40, 44, 45, 48, 51, 116).

Wang, Yulong, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu (2020b). "Pruning from scratch". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07, pp. 12273–12280 (cit. on p. 52).

Xie, Yueqi, Ka Leong Cheng, and Qifeng Chen (2021). "Enhanced invertible encoding for learned image compression". In: *Proceedings of the 29th ACM International Conference on Multimedia*, pp. 162–170 (cit. on p. 86).

Xie, Yujia, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister (2020). "Differentiable top-k with optimal transport". In: *Advances in Neural Information Processing Systems* 33, pp. 20520–20531 (cit. on p. 90).

Xue, Tianfan, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman (2019). "Video enhancement with task-oriented flow". In: *International Journal of Computer Vision* 127.8, pp. 1106–1125 (cit. on pp. 84, 133).

Yang, Yang, Guillaume Sautiére, J. Jon Ryu, and Taco S Cohen (2019). "Feedback Recurrent AutoEncoder". In: *IEEE International Conference on Acoustics, Speech and Signal Processing* (cit. on p. 87).

Yang, Yibo, Robert Bamler, and Stephan Mandt (2020). "Improving Inference for Neural Image Compression". In: *NeurIPS* (cit. on pp. 86, 87).

Ye, Shaokai, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin (2019). "Adversarial robustness vs. model compression, or both?" In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 111–120 (cit. on p. 11).

Yin, Penghang, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin (2018). "Binaryrelax: A relaxation approach for training deep neural networks with

quantized weights". In: *SIAM Journal on Imaging Sciences* 11.4, pp. 2205–2223 (cit. on p. 67).

You, Haoran, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Yingyan Lin, Zhangyang Wang, and Richard G. Baraniuk (2020). "Drawing Early-Bird Tickets: Toward More Efficient Training of Deep Networks". In: *International Conference on Learning Representations* (cit. on pp. 35, 52).

Yu, Alex, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa (2020). "pixelNeRF: Neural Radiance Fields from One or Few Images". In: *arXiv:2012.02190 [cs]* (cit. on p. 87).

Zbontar, Jure, Florian Knoll, Anuroop Sriram, Tullie Murrell, Zhengnan Huang, Matthew J Muckley, Aaron Defazio, Ruben Stern, Patricia Johnson, Mary Bruno, et al. (2018). "fastMRI: An open dataset and benchmarks for accelerated MRI". In: *arXiv preprint arXiv:1811.08839* (cit. on pp. 85, 133, 134).

Zeghidour, Neil, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi (2021). "Soundstream: An end-to-end neural audio codec". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* (cit. on p. 87).

Zhang, Susan, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer (2022). *OPT: Open Pre-trained Transformer Language Models* (cit. on p. 2).

Zhang, Yunfan, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen (2021). "Implicit Neural Video Compression". In: *arXiv preprint arXiv:2112.11312* (cit. on p. 88).

Zhao, Ritchie, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang (2019). "Improving Neural Network Quantization without Retraining using Outlier Channel Splitting". In: *International Conference on Machine Learning*, pp. 7543–7552 (cit. on p. 54).

Zhou, Shuchang, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou (2016). "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients". In: *CoRR* abs/1606.06160 (cit. on p. 16).

Zhu, Michael H. and Suyog Gupta (2018). *To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression* (cit. on pp. 14, 35, 52).

Zintgraf, Luisa, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson (2019a). "Fast context adaptation via meta-learning". In: *International Conference on Machine Learning*. PMLR, pp. 7693–7702 (cit. on p. 76).

Zintgraf, Luisa, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson (June 2019b). "Fast Context Adaptation via Meta-Learning". In: *ICML 2019: Proceedings of the Thirty-Sixth International Conference on Machine Learning* (cit. on p. 21).

Appendix A

Appendix to Chapter 3

A.1 3SP Experimental Details

For VGG-19 we use an architecture of five blocks, with 2x2 max-pooling layers in between the blocks. The first two blocks consist of two conv-BN-relu operations, with 64 and 128 channels, respectively. The last three blocks consist of three conv-BN-relu operations and are of width 256, 512 and 512. After our model we use average pooling to reduce spatial dimensions to 2 by 2. This is followed by three linear layers with 1024, 512, and number of classes as number of hidden units. We use ReLU activation functions throughout the model. During structured pruning, we consider all elements of the model prunable except the output of the final linear layer and the input of the first convolution layer. We train our model for a fixed 160 epochs, with an initial learning rate of 0.1, which is halved at epoch 80 and again at 120. We preprocess CIFAR-10 by doing mean and standard deviation normalization, random horizontal flips, and random 4-pixel pads followed by a 32x32 crop. We report results on the CIFAR-10 test set. This is the same experimental setup as used in Wang et al. (2020a); we did not consider other hyperparameters. We use the default initialization method of Pytorch 1.4 for the weights, which is based on He et al. (2015). All results are based on 5 training/evaluation runs with different random seeds.

For all full model measurements of computational cost in this paper, we utilize

the `pytorch-OpCounter`[1] package, which takes into account all operations of the model, excluding preprocessing.

A.2 3SP Wall Clock Time and Model Size Experiments

We performed the experiments in Table A.1 using the VGG-19 described in Appendix A.1. The training time per epoch was measured as the average of 5 epochs, excluding the first epoch. The prediction time is the average prediction time per data point after going through an epoch of data one by one. The model size is determined by saving the parameters to disk. Pruning time is determined by doing a forward pass with a single batch of data in the original model. The time necessary for adjusting the architecture is excluded but would add some overhead (under 10 ms) in practice. The 50% FLOPs reduction model is 3SP without compute aware. The 80% FLOPs reduction model is 3SP with compute aware.

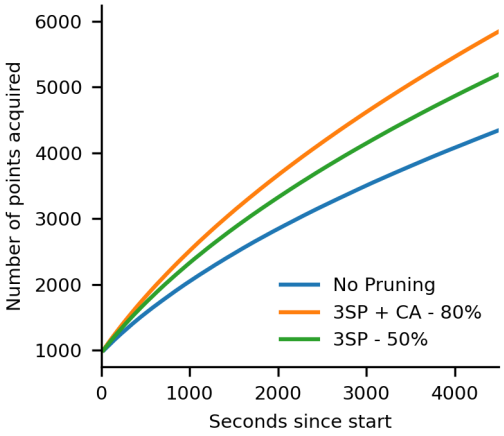
Table A.1: Effect of the computational cost reduction on training/inference time. Reducing FLOPs directly reduces training time and prediction time, and indirectly reduces the model size. These results are obtained using the ResNet-18 model, CIFAR10 dataset, and a RTX 2080 Ti GPU. The prediction time is measured using a batch with a single element on 4 Intel Xeon E5-2680 cores.

Model	GPU Train (s/epoch)	CPU Predict (ms)	Size (MB)
Original	24.94	23.6	85.3
SSSP (65% FLOPS)	11.44	10.84	6.31
SSSP + CA (75% FLOPS)	9.53	8.74	6.32

A.3 3SP Active Learning Experiments

In active learning (Settles, 2010), we want to be efficient about manually labeling data and only acquire labels for the most informative data points. It is also referred to as a 'human in the loop' approach to data labeling. One begins with a small training dataset, and obtains labels only for the most informative datapoints in

[1]Available from https://github.com/Lyken17/pytorch-OpCounter



r

Figure A.1: Active Learning CIFAR-10. 3SP model at 50% and 3SP-CA at 80% FLOPs reduction get to acquire more data within a time-budget because they train faster.

the unlabeled set. In our case, we start with 100 examples per class, leading to a start dataset of size 1000 for CIFAR-10. We train the model on the current labelled dataset and subsequently use it to estimate which datapoints in the original CIFAR-10 training set (the 'unlabeled set') would be most informative to acquire a label for. We use a common proxy for informativeness: the entropy of the softmax distribution $H(y) = \sum_i p(y_i|\mathbf{x}) \log p(y_i|\mathbf{x})$, where $p(y|\mathbf{x})$ represents the output distribution for a particular data point $\mathbf{x}$ (Gal et al., 2017). We select 50 datapoints from the unlabeled pool in each acquisition step, and add these (including labels) to the training set and restart training (using the final state of the model before acquiring more data). We repeat this process until the time runs out. We obtain data from the CIFAR-10 training set and report results on the CIFAR-10 test set. We prune the models once at the beginning of the active learning process. We start the timer after we acquired the 1000 initial points and are done with pruning. The experiments were done using the VGG-19 architecture and training settings described in Appendix A.1. For 3SP - 50%, we pruned 50% of the FLOPS of the model, without compute awareness. For 3SP + CA - 80%, we pruned 80% of the FLOPS, but with compute awareness. We repeat the experiments 5 times, and show one standard deviation errors.

Throughout active learning, the pruned models train more quickly. This means that they are able to select points for inclusion more quickly (see Figure A.1). 3SP + CA sees almost 50% more data than the un-pruned model within the time-budget.

A.4 ProsPr Experimental Details

A.4.1 ProsPr Architectures

We use standard VGG and ResNet models provided by `torchvision` throughout this work where possible. The ResNet-20 model, which is not commonly evaluated, was implemented to match the version used by Frankle et al. (2021) so that we could compare using the benchmark supplied by this paper.

For smaller datasets, it is common to patch models defined for ImageNet. Specifically, for ResNets, we replace the first convolution with one 3×3 filter size, and stride 1; the first max-pooling layer is replaced with an identity operation. For VGG, we follow the convention used by works such as FORCE (Jorge et al., 2021). We do not change any convolutional layers, but we change the classifier to use a single global average pooling layer, followed by a single fully-connected layer.

A.4.2 ProsPr Training Details

For CIFAR-10, CIFAR-100 and TinyImageNet we perform 3 meta-steps to calculate our saliency criteria. We train the resulting models for 200 epochs, with initial learning rate 0.1; we divide the learning rate by 10 at epochs 100 and 150. Weight decay was set to 5×10^{-4}. Batch size for CIFAR-10, CIFAR-100, and TinyImageNet was 256. For CIFAR-10 and CIFAR-100 we augment training data by applying random cropping (32×32, padding 4), and horizontal flipping. For TinyImageNet we use the same procedure, with random cropping parameters set to 64×64, padding 4.

For ImageNet we train models for 100 epochs, with an initial learning rate of 0.1; we divide the learning rate by 10 at epochs 30, 60 and 90. Weight decay was set to 1×10^{-4}. Batch size was 256. We use the first order approximation to do pruning, and use 1024 steps for ResNet-50. For VGG-19 we use 2048 steps, but with batch size set to 128 (due to memory limitations, as our implementation only utilized a single GPU for meta-training). We apply random resizing, then crop the image to 224×224, with horizontal flipping.

A.4.3 Implementations

In addition to our code, the reader may find it useful to reference the following repos from related work. Our experiments were performed using code derived from these implementations:

- https://github.com/naver/force
- https://github.com/alecwangcq/GraSP
- https://github.com/facebookresearch/open_lth
- https://github.com/ganguli-lab/Synaptic-Flow
- https://github.com/mil-ad/snip

A.5 ProsPr Numbers from Figure 3.9

The numbers used to produce Figure 3.9 are included in Tables A.2, A.3, and A.4. These Tables are placed on the next page to make better use of space.

Table A.2: Numerical results for ResNet-20 on CIFAR-10

Sparsity (%)	20.0	36.0	48.8	59.0	67.2	73.8	79.0	83.2	86.6	89.3	91.4	93.1	94.5	95.6	96.5	97.2	97.7	98.2
LTR after Training	91.8 ± 0.2	91.9 ± 0.2	91.9 ± 0.2	91.7 ± 0.2	91.5 ± 0.1	91.4 ± 0.1	91.1 ± 0.1	90.6 ± 0.1	90.1 ± 0.0	89.2 ± 0.1	88.0 ± 0.2	86.8 ± 0.2	85.7 ± 0.1	84.4 ± 0.2	82.8 ± 0.1	81.2 ± 0.3	79.4 ± 0.3	77.3 ± 0.5
Magnitude after Training	92.2 ± 0.3	92.0 ± 0.2	92.0 ± 0.2	91.7 ± 0.1	91.5 ± 0.2	91.3 ± 0.2	91.1 ± 0.2	90.7 ± 0.2	90.2 ± 0.2	89.4 ± 0.2	88.7 ± 0.2	87.7 ± 0.2	86.5 ± 0.2	85.2 ± 0.2	83.5 ± 0.3	81.9 ± 0.3	80.4 ± 0.2	77.7 ± 0.4
Magnitude at Initialization	91.5 ± 0.2	91.2 ± 0.1	90.8 ± 0.1	90.7 ± 0.2	90.2 ± 0.1	89.8 ± 0.2	89.3 ± 0.2	88.6 ± 0.2	87.9 ± 0.3	87.0 ± 0.3	86.1 ± 0.2	85.2 ± 0.4	83.9 ± 0.2	82.5 ± 0.4	80.7 ± 0.5	79.1 ± 0.4	77.2 ± 0.4	74.5 ± 0.7
SNIP	91.8 ± 0.2	91.2 ± 0.3	90.9 ± 0.1	90.7 ± 0.1	90.1 ± 0.2	89.7 ± 0.3	89.0 ± 0.2	88.5 ± 0.3	87.7 ± 0.2	87.2 ± 0.4	85.8 ± 0.1	84.7 ± 0.5	83.8 ± 0.3	82.5 ± 0.4	80.9 ± 0.2	79.1 ± 0.2	77.3 ± 0.2	74.0 ± 0.5
GraSP	91.5 ± 0.1	91.3 ± 0.2	91.2 ± 0.1	90.6 ± 0.2	90.3 ± 0.2	89.6 ± 0.1	89.1 ± 0.2	88.4 ± 0.2	87.9 ± 0.1	87.0 ± 0.2	85.9 ± 0.1	85.1 ± 0.4	83.9 ± 0.4	82.8 ± 0.2	81.2 ± 0.2	79.7 ± 0.3	78.0 ± 0.3	76.0 ± 0.5
SynFlow	91.7 ± 0.1	91.3 ± 0.2	91.2 ± 0.1	90.8 ± 0.1	90.4 ± 0.2	89.8 ± 0.1	89.5 ± 0.3	88.9 ± 0.4	88.1 ± 0.1	87.4 ± 0.5	86.1 ± 0.2	85.4 ± 0.2	84.3 ± 0.2	82.9 ± 0.2	81.7 ± 0.2	80.0 ± 0.3	78.6 ± 0.4	76.4 ± 0.4
Random	91.6 ± 0.2	91.2 ± 0.2	90.8 ± 0.3	90.5 ± 0.2	89.8 ± 0.2	89.0 ± 0.4	88.4 ± 0.2	87.5 ± 0.3	86.6 ± 0.2	85.6 ± 0.3	84.3 ± 0.4	83.1 ± 0.4	81.6 ± 0.3	79.6 ± 0.4	74.2 ± 6.4	64.7 ± 9.7	56.9 ± 8.5	43.7 ± 12.5
ProsPr	92.3 ± 0.1	92.1 ± 0.0	91.7 ± 0.2	91.5 ± 0.1	91.0 ±0.2	90.5 ± 0.0	90.1 ± 0.1	89.6 ± 0.2	88.5 ± 0.5	87.8 ± 0.1	86.9 ± 0.3	85.5 ± 0.6	84.3 ± 0.2	83.0 ± 0.9	80.8 ± 0.5	79.6 ± 0.7	77.0 ± 0.8	74.2 ± 0.3

Table A.3: Numerical results for VGG-16 on CIFAR-10

Sparsity (%)	20.0	36.0	48.8	59.0	67.2	73.8	79.0	83.2	86.6	89.3	91.4	93.1	94.5	95.6	96.5	97.2	97.7	98.2
LTR after Training	93.5 ± 0.1	93.6 ± 0.1	93.6 ± 0.1	93.6 ± 0.1	93.8 ± 0.1	93.6 ± 0.1	93.6 ± 0.1	93.8 ± 0.1	93.8 ± 0.1	93.7 ± 0.1	93.7 ± 0.1	93.8 ± 0.1	93.5 ± 0.2	93.4 ± 0.1	93.2 ± 0.1	93.0 ± 0.2	92.7 ± 0.1	92.1 ± 0.4
Magnitude after Training	93.9 ± 0.2	93.9 ± 0.2	93.8 ± 0.1	93.8 ± 0.1	93.9 ± 0.1	94.0 ± 0.2	93.8 ± 0.1	93.8 ± 0.1	93.9 ± 0.2	93.9 ± 0.2	93.8 ± 0.2	93.7 ± 0.2	93.5 ± 0.1	93.5 ± 0.1	93.3 ± 0.2	93.0 ± 0.1	92.9 ± 0.1	91.7 ± 0.8
Magnitude at Initialization	93.6 ± 0.2	93.4 ± 0.2	93.3 ± 0.1	93.2 ± 0.2	93.3 ± 0.3	93.0 ± 0.1	93.1 ± 0.1	92.9 ± 0.1	92.9 ± 0.2	92.7 ± 0.1	92.5 ± 0.2	92.3 ± 0.1	92.2 ± 0.2	92.0 ± 0.1	91.8 ± 0.2	91.5 ± 0.1	91.3 ± 0.3	90.9 ± 0.2
SNIP	93.6 ± 0.1	93.4 ± 0.1	93.3 ± 0.1	93.4 ± 0.2	93.3 ± 0.2	93.4 ± 0.1	93.1 ± 0.1	93.1 ± 0.1	93.2 ± 0.1	93.1 ± 0.1	92.9 ± 0.1	92.8 ± 0.2	92.8 ± 0.1	92.3 ± 0.2	92.2 ± 0.1	92.1 ± 0.1	91.7 ± 0.1	91.5 ± 0.1
GraSP	93.5 ± 0.1	93.4 ± 0.2	93.5 ± 0.0	93.3 ± 0.1	93.2 ± 0.2	93.3 ± 0.2	93.2 ± 0.1	93.0 ± 0.3	93.0 ± 0.1	92.7 ± 0.2	92.8 ± 0.1	92.4 ± 0.1	92.3 ± 0.1	92.2 ± 0.1	91.9 ± 0.1	91.6 ± 0.2	91.5 ± 0.0	91.2 ± 0.2
SynFlow	93.6 ± 0.2	93.6 ± 0.1	93.5 ± 0.1	93.4 ± 0.1	93.4 ± 0.2	93.5 ± 0.2	93.2 ± 0.1	93.2 ± 0.1	93.1 ± 0.1	92.9 ± 0.1	92.7 ± 0.2	92.5 ± 0.1	92.3 ± 0.1	92.0 ± 0.1	91.8 ± 0.3	91.3 ± 0.1	91.0 ± 0.2	90.6 ± 0.2
Random	93.6 ± 0.3	93.2 ± 0.1	93.2 ± 0.2	93.0 ± 0.2	92.7 ± 0.2	92.4 ± 0.2	92.2 ± 0.1	91.7 ± 0.1	91.2 ± 0.1	90.8 ± 0.2	90.3 ± 0.2	89.6 ± 0.2	88.8 ± 0.2	88.3 ± 0.4	87.6 ± 0.1	86.4 ± 0.2	86.0 ± 0.4	84.5 ± 0.4
ProsPr	93.7 ± 0.2	93.7 ± 0.1	93.9 ± 0.1	93.8 ± 0.1	93.8 ± 0.1	93.5 ± 0.2	93.6 ± 0.1	93.4 ± 0.3	93.5 ± 0.2	93.3 ± 0.1	93.0 ± 0.1	93.0 ± 0.1	92.8 ± 0.3	92.7 ± 0.1	92.6 ± 0.1	92.2 ± 0.1	92.1 ± 0.2	91.6 ± 0.4

Table A.4: Numerical results for ResNet-18 on TinyImageNet

Sparsity (%)	20.0	36.0	48.8	59.0	67.2	73.8	79.0	83.2	86.6	89.3	91.4	93.1	94.5	95.6	96.5	97.2	97.7	98.2
LTR after Training	51.7 ± 0.2	51.4 ± 0.3	51.5 ± 0.4	52.1 ± 0.4	51.8 ± 0.4	52.0 ± 0.1	52.0 ± 0.1	52.0 ± 0.2	52.1 ± 0.3	52.0 ± 0.2	52.4 ± 0.2	51.8 ± 0.4	51.8 ± 0.6	51.4 ± 0.4	50.9 ± 0.2	49.3 ± 0.7	48.3 ± 0.7	46.0 ± 0.3
Magnitude after Training	51.7 ± 0.3	51.4 ± 0.1	51.7 ± 0.2	51.5 ± 0.3	51.7 ± 0.4	51.4 ± 0.5	51.1 ± 0.3	51.4 ± 0.4	51.3 ± 0.4	51.1 ± 0.6	51.7 ± 0.3	51.3 ± 0.3	51.8 ± 0.4	51.2 ± 0.3	51.1 ± 0.2	50.4 ± 0.2	49.0 ± 0.2	47.8 ± 0.5
Magnitude at Initialization	51.0 ± 0.3	51.2 ± 0.3	51.0 ± 0.2	50.5 ± 0.5	50.6 ± 0.3	50.0 ± 0.3	50.3 ± 0.2	50.3 ± 0.3	50.0 ± 0.1	49.8 ± 0.5	49.0 ± 0.1	48.3 ± 0.3	47.2 ± 0.2	46.2 ± 0.2	44.4 ± 0.5	42.2 ± 0.1	40.8 ± 0.4	38.1 ± 0.6
SNIP	51.4 ± 0.2	51.5 ± 0.3	51.4 ± 0.3	51.3 ± 0.5	51.6 ± 0.4	51.4 ± 0.5	51.9 ± 0.6	51.5 ± 0.3	51.0 ± 0.2	51.2 ± 0.7	50.6 ± 0.3	50.1 ± 0.3	49.2 ± 0.3	47.8 ± 0.2	46.7 ± 0.1	45.2 ± 0.4	44.5 ± 0.3	42.3 ± 0.3
GraSP	49.8 ± 0.4	49.1 ± 0.3	49.5 ± 0.2	49.5 ± 0.4	49.2 ± 0.1	49.5 ± 0.2	48.7 ± 0.1	49.0 ± 0.5	48.8 ± 0.4	48.3 ± 0.1	48.2 ± 0.1	47.7 ± 0.2	46.5 ± 0.1	45.5 ± 0.7	44.9 ± 0.2	44.1 ± 1.0	42.9 ± 0.5	41.0 ± 0.1
SynFlow	51.8 ± 0.3	51.6 ± 0.3	51.7 ± 0.7	51.8 ± 0.2	51.3 ± 0.4	51.3 ± 0.4	51.5 ± 0.2	51.0 ± 0.4	50.2 ± 0.4	50.4 ± 0.3	49.1 ± 0.0	48.0 ± 0.5	46.7 ± 0.7	45.6 ± 0.0	44.0 ± 0.2	42.2 ± 0.3	40.0 ± 0.1	38.2 ± 0.5
Random	50.6 ± 0.5	50.1 ± 0.2	49.9 ± 0.3	48.7 ± 0.2	48.0 ± 0.4	48.0 ± 0.6	46.4 ± 0.1	45.9 ± 0.5	44.7 ± 0.2	43.6 ± 0.3	42.7 ± 0.2	41.4 ± 0.4	40.2 ± 0.2	37.2 ± 0.2	36.2 ± 0.7	34.0 ± 0.4	32.2 ± 0.5	30.0 ± 0.3
ProsPr	51.8 ± 0.4	51.4 ± 0.7	51.2 ± 0.9	52.0 ± 0.2	51.8 ± 0.1	51.2 ± 0.4	52.0 ± 0.3	51.6 ± 0.7	51.1 ± 0.4	50.7 ± 0.6	50.9 ± 0.3	50.8 ± 1.2	51.1 ± 0.7	50.8 ± 0.5	50.8 ± 0.3	49.6 ± 0.6	49.2 ± 0.2	46.9 ± 0.7

A.6 ProsPr Wall Clock Time for Structured Pruning

When pruning is done at convergence, the benefits of having a compressed model (in terms of *memory saving* and *speed-up*) can only be utilized at inference/deployment time. However, with pruning-at-initialization these benefits can be reaped during training as well. This is especially true in the case of structured pruning, where pruning results in weight and convolutional kernels with smaller dimensions (as opposed to unstructured pruning, where we end up with sparse weights with the original dimensions). This means that in addition to memory savings, training take fewer operations which speeds up training. To evaluate the benefits of training at initialization in terms of speed improvements we measured the wall-time training time on an NVIDIA RTX 2080 Ti GPU for the architectures used in Section 3.4.2 (an additionally on ImageNet dataset). The results in Table A.5 show that structured pruning with ProsPr can significantly reduce the overall training time.

Table A.5: Wall-time Training Speed-ups by Structured Pruning at Initialization

Dataset	Epochs	Model	Unpruned	80% pruned	90% pruned	95% pruned
CIFAR-100	200	ResNet-18	83.9 mins	60.76 mins	54.8 mins	46.6 mins
CIFAR-100	200	VGG-19	50.3 mins	45.8 mins	39.93 mins	38.8 mins
Tiny-ImageNet	200	ResNet-18	9.79 hours	8 hours	5.4 hours	4.92 hours
Tiny-ImageNet	200	VGG-19	5.75 hours	4.0 hours	3.38 hours	2.7 hours
ImageNet	90	ResNet-18	73.7 hours	72.15 hours	65.9 hours	64.6 hours

A.7 ProsPr Results on Segmentation Task

An interesting, albeit less common, application for pruning models is within the context of segmentation. In a recent paper Jeong et al. (2021) train and prune the U-Net (Ronneberger et al., 2015) architecture on two image datasets from the Cell Tracking Challenge (PhC-C2DH-U373 and DIC-C2DH-HeLa). They use the classic multi-step approach of gradually applying magnitude-pruning interleaved with fine-tuning stages. To evaluate the flexibility of our method we used meta-gradients at the beginning of training (on a randomly initialized U-Net), prune in a single shot, and train the network once for the same number of epochs (50). We kept

the training set-up the same as the baseline by Jeong et al. (2021) (i.e., resizing images and segmentation maps to (256,256), setting aside 30% of training data for validation) and similarly aim to find the highest prune ratio that does not result in IOU degradation. We report intersection-over-union (IOU) metric for the two datasets in Tables A.6 and A.7:

Table A.6: Mean-IOU on U373 validation

Method	Prune Ratio	Mean IOU
Unpruned	-	0.9371
Jeong et al.	95%	0.9368
ProsPr	97%	0.9369

Table A.7: Mean-IOU on HeLa validation

Method	Prune Ratio	Mean IOU
Unpruned	-	0.7514
Jeong et al.	81.8%	0.7411
ProsPr	90%	0.7491

These results show that our method works as well (or better) compared to this compute-expensive baseline, in the sense that we can prune more parameters while keeping the IOU score the same.

A.8 ProsPr Self-supervised Initialization

To evaluate the robustness and consistency of our method against non-random initialization we ran experiments using BYOL to learn representations from unlabeled samples (Grill et al., 2020). We used ResNet-18 as a backbone and trained for 1000 epochs with an embedding size of 64. Unlike the vanilla ResNet-18 architecture used in Section 3.4.2 we used the commonly-used modified version of ResNet-18 for smaller inputs (removing the first pooling layer and modifying the first convolutional layer to have kernel kernel size of 3, stride of 1, and padding size of 1). We then used this trained ResNet18 as the initialization for our meta-gradient pruning method. After the pruning step, all layers were trained as before until convergence. All training hyper-parameters were kept as before. The results (final test accuracies for 95% pruning) are summarized in Table A.8.

These results show the robustness of our method for this particular self-supervised initialization. Starting from a learned representation can be challenging because these representations are much closer to weight values at convergence, and therefore the magnitude of their gradients is significantly smaller than randomly initialized weights. However, this is less of a problem for meta-gradients as their magnitude is

Table A.8: Comparing test accuracies (%) of ProsPr on randomly initialized ResNet-18 and initialization from self-supervised learning (BYOL)

Dataset	Random Init	BYOL init
CIFAR-10	93.6	93.62
CIFAR-100	73.2	74.02

still significant due to back-propagation through training steps. This can be seen in Figure A.2 which shows the L2 norm of gradients of each layer of a BYOL-initialized ResNet-18 for meta-gradients compared to normal gradients. It can be seen that meta-gradients provide a stronger signal compared to normal gradients.

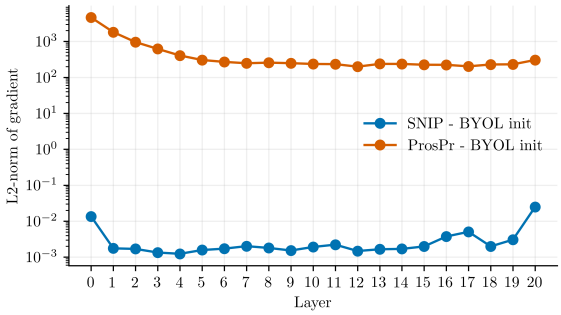


Figure A.2: Comparing L2-norm of gradients and meta-gradients in BYOL-initialized ResNet-18

Appendix B

Appendix to Chapter 4

B.1 Robustness Analysis for Quantization perturbations

In this section, we address two questions in more details, first regarding regularization of the ℓ_2-norm of gradient and second regarding non-uniform quantization schemes.

We argued above that regularizing the ℓ_2-norm of gradient cannot achieve the same level of robustness as regularization of the ℓ_1-norm of gradient. We provide here another, more theoretical, argument. The following inequality shows how the ℓ_2-norm of gradient controls the first-order perturbation:

$$\langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w}) \rangle \leq \|\boldsymbol{\Delta}\|_2 \|\nabla f(\boldsymbol{w})\|_2 .$$

This is a simple Cauchy-Shwartz inequality. Therefore, if the ℓ_2-norm of the gradient is inversely proportional to the *power* of the perturbation, the first-order term is adequately controlled. However, using a theoretical argument, we show that the *power* of the ℓ_∞-bounded perturbation can blow up with the dimension as a vector $\boldsymbol{\Delta}$ in $\mathbb{R}^n$ with $\|\boldsymbol{\Delta}\|_\infty = \delta$ can reach an ℓ_2-norm of approximately $\sqrt{n}\delta$. In other words, the length of the quantization noise behaves with high probability as $\Theta(\sqrt{n})$, which implies that the the ℓ_2-norm of the gradient should be as small as $\Theta(1/\sqrt{n})$.

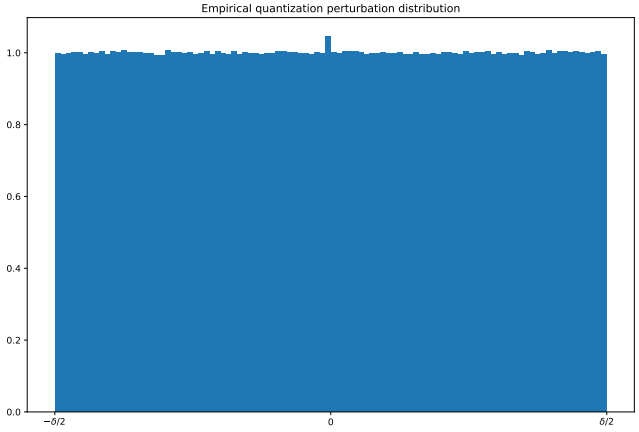


Figure B.1: Quantization noise is uniformly distributed. In this plot we show the quantization noise on each individual weight in an ImageNet trained ResNet18 model. The noise is scaled by the width of the quantization bin for each weight quantizer. This plot shows that quantization noise is uniformly distributed between $-\delta/2$ and $\delta/2$.

We show that this can indeed occur with high probability for any random quantization noise with the bounded support. Note that for symmetric uniform quantization schemes, quantization noise can be approximated well by a uniform distribution over $[-\delta/2, \delta/2]$ where δ is the width of the quantization bin. See Figures B.1 for the empirical distribution of quantization noise on the weights of a trained network. Our argument, however, works for any distribution supported over $[-\delta/2, \delta/2]$, and, therefore, it includes asymmetric quantization schemes over a uniform quantization bin.

Consider a vector $\boldsymbol{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ with entries x_i randomly and independently drawn from a distribution supported on $[-\delta/2, \delta/2]$. We would like to show that $\|\boldsymbol{x}\|_2^2$ is well concentrated around its expected values. To do that we are going to write down the above norm as the sum of independent zero-mean random variables. See that:

$$\mathbb{E}\left(\|\boldsymbol{x}\|_2^2\right) = \mathbb{E}\left(\sum_{i=1}^{n} x_i^2\right) = n\mathbb{E}\left(x_1^2\right) = \frac{n\delta^2}{12}.$$

Besides, note that $x_i^2 \in [0, \delta^2/4]$. Therefore $x_i^2 - \delta^2/12$ is a zero-mean random variable that lies in the interval $[-\delta^2/12, \delta^2/6]$. We can now use Hoeffding's inequality. To

be self-contained, we include the theorem below.

Theorem B.1.1 (Hoeffding's inequality, (Hoeffding, 1963)) *Let $X_1, \ldots, X_n$ be a sequence of independent zero-mean random variables such that X_i is almost surely supported on $[a_i, b_i]$ for $i \in \{1, \ldots, n\}$. Then, for all $t > 0$, it holds that*

$$\mathbb{P}\left(\sum_{i=1}^{n} X_i \geq t\right) \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{B.1}$$

$$\mathbb{P}\left(\left|\sum_{i=1}^{n} X_i\right| \geq t\right) \leq 2\exp\left(-\frac{t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{B.2}$$

Applying Theorem B.1.1 to our setting, we obtain:

$$\mathbb{P}\left(\left|\|\boldsymbol{x}\|_2^2 - n\delta^2/12\right| \geq t\right) \leq 2\exp\left(-\frac{2t^2}{n(\delta^2/4)^2}\right).$$

So with probability $1 - \epsilon$, we have:

$$\left|\|\boldsymbol{x}\|_2^2 - n\delta^2/12\right| \leq \left(\frac{n\delta^4}{32}\log(2/\epsilon)\right)^{1/2}.$$

Therefore, if the quantization noise $\boldsymbol{\Delta}$ has entries randomly drawn from a distribution over $[-\delta/2, \delta/2]$, then with probability $1 - \epsilon$, the squared ℓ_2-norm of $\boldsymbol{\Delta}$, i.e., $\|\boldsymbol{\Delta}\|_2^2$, lies in the interval $\left[\frac{n\delta^2}{12} - \sqrt{\frac{n\delta^4}{32}\log(2/\epsilon)}, \frac{n\delta^2}{12} + \sqrt{\frac{n\delta^4}{32}\log(2/\epsilon)}\right]$. In other words, the length of the vector behaves with high probability as $\Theta(\sqrt{n})$. This result holds for any quantization noise with bounded support.

If the quantization bins are non-uniformly chosen, and if the weights can take arbitrarily large values, the quantization noise is no-longer bounded in general. As long as the quantization noise has a Gaussian tail, i.e., it is a subgaussian random variable, one can use Hoeffding's inequality for subgaussian random variables to show a similar concentration result as above. The *power* of the perturbation will, therefore, behave with $\Theta(\sqrt{n})$, and the ℓ_2-norm of the gradient cannot effectively control the gradient. Note that nonuniform quantization schemes are not commonly used for hardware implementations, hence, our focus on uniform cases. Besides, the validity of this assumption about nonuniform quantization noise requires further

investigation, which is relegated to our future works.

B.2 Second-Order Perturbation Analysis

We start by writing the approximation of $f(\cdot)$ up to the second-order term:

$$f(\boldsymbol{w}+\boldsymbol{\Delta}) = f(\boldsymbol{w}) + \langle \boldsymbol{\Delta}, \nabla f(\boldsymbol{w})\rangle + \frac{1}{2}\boldsymbol{\Delta}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{\Delta} + R_3.$$

The worst-case second-order term under ℓ_∞-bounded perturbations is given by

$$\max_{\|n\|_\infty \le \delta} \boldsymbol{\Delta}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{\Delta}.$$

The above value is difficult to quantify for general case. We demonstrate this difficulty by considering some special cases.

Let's start with convex functions, for which the Hessian $\nabla^2 f(\boldsymbol{w})$ is positive semi-definite. In this case, the Hessian matrix admits a square root, and the second-order term can be written as:

$$\boldsymbol{\Delta}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{\Delta} = \boldsymbol{\Delta}^T(\nabla^2 f(\boldsymbol{w}))^{1/2}(\nabla^2 f(\boldsymbol{w}))^{1/2}\boldsymbol{\Delta} = \left\|(\nabla^2 f(\boldsymbol{w}))^{1/2}\boldsymbol{\Delta}\right\|_2^2.$$

Therefore the worst-case analysis of the second-term amounts to

$$\max_{\|n\|_\infty \le \delta} \boldsymbol{\Delta}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{\Delta} = \max_{\|n\|_\infty \le \delta} \left\|(\nabla^2 f(\boldsymbol{w}))^{1/2}\boldsymbol{\Delta}\right\|_2^2.$$

The last term is the mixed $\infty \to 2$-norm of $(\nabla^2 f(\boldsymbol{w}))^{1/2}$. As a reminder, the $p \to q$-matrix norm is defined as

$$\|\boldsymbol{A}\|_{p\to q} := \max_{\|x\|_p \le 1} \|\boldsymbol{A}\|_q = \max_{\substack{\|\boldsymbol{x}\|_p \le 1 \\ \|\boldsymbol{y}\|_{q^*} \le 1}} \langle \boldsymbol{y}, \boldsymbol{A}\boldsymbol{x}\rangle =: \left\|\boldsymbol{A}^T\right\|_{q^*\to p^*}$$

where p^*, q^* denote the dual norms of p and q, i.e. satisfying $1/p+1/p^* = 1/q+1/q^* = 1$.

The worst case second-order perturbation is given by:

$$\max_{\|n\|_\infty \le \delta} \boldsymbol{\Delta}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{\Delta} = \delta^2 \left\|(\nabla^2 f(\boldsymbol{w}))^{1/2}\right\|_{\infty\to 2}^2.$$

Unfortunately the $\infty \to 2$-norm is known to be NP-hard ((Hendrickx et al., 2010); see Bhattiprolu et al. (2018) for a more recent study). As a matter of fact, if $f(\cdot)$ is positive semi-definite, and hence the function is convex, the problem above corresponds to maximization of convex functions, which is difficult as well.

For a general Hessian, the problem is still difficult to solve. First note that:

$$\max_{\|n\|_\infty \leq \delta} \boldsymbol{\Delta}^T \nabla^2 f(\boldsymbol{w}) \boldsymbol{\Delta} = \max_{\|n\|_\infty \leq \delta} \operatorname{Tr}\left(\nabla^2 f(\boldsymbol{w}) \boldsymbol{\Delta}\boldsymbol{\Delta}^T\right).$$

We can therefore replace $\boldsymbol{\Delta}\boldsymbol{\Delta}^T$ with a positive semi-defintite matrix of rank 1 denoted by $\boldsymbol{N}$. The worst case second-order perturbation can be obtained by solving the following problem:

$$\begin{aligned} \max_{\boldsymbol{N} \in \mathbb{R}^{n \times n}} \ & \operatorname{Tr}\left(\nabla^2 f(\boldsymbol{w}) \boldsymbol{N}\right) \\ \text{subject to } & \boldsymbol{N} \succeq 0 \\ & N_{ii} \leq \delta^2 \quad \text{for } i \in \{1, \ldots, n\} \\ & \operatorname{rank}(\boldsymbol{N}) = 1. \end{aligned} \tag{B.3}$$

The last constraint, namely the rank constraint, is a discrete constraint. The optimization problem above is therefore NP-hard to solve. To sum up, the worst case second-order perturbation cannot be efficiently computed, which poses difficulty for controlling the second-order robustness.

There are, however, approximations available in the literature. A common approximation, which is widely known for the Max-Cut and community detection problems, consists of dropping the rank-constraint from the above optimization problem to get the following semi-definite program:

$$\begin{aligned} \max_{\boldsymbol{N} \in \mathbb{R}^{n \times n}} \ & \operatorname{Tr}\left(\nabla^2 f(\boldsymbol{w}) \boldsymbol{N}\right) \\ \text{subject to } & \boldsymbol{N} \succeq 0 \\ & N_{ii} \leq \delta^2 \quad \text{for } i \in \{1, \ldots, n\} \end{aligned} \tag{B.4}$$

Unfortunately this approximation, apart from being costly to solve for large n, does not provide a regularization parameter that can be included in the training of the model.

It is not clear how we can control the second-order term through a tractable term.

B.3 DQ Imposes 4th Power Constraint on Singular Values

From basic linear algebra we have that

$$\|\boldsymbol{W}\|_2^2 = \mathrm{Tr}(\boldsymbol{W}^T\boldsymbol{W}) = \sum_i \sigma_i^2(\boldsymbol{W}),$$

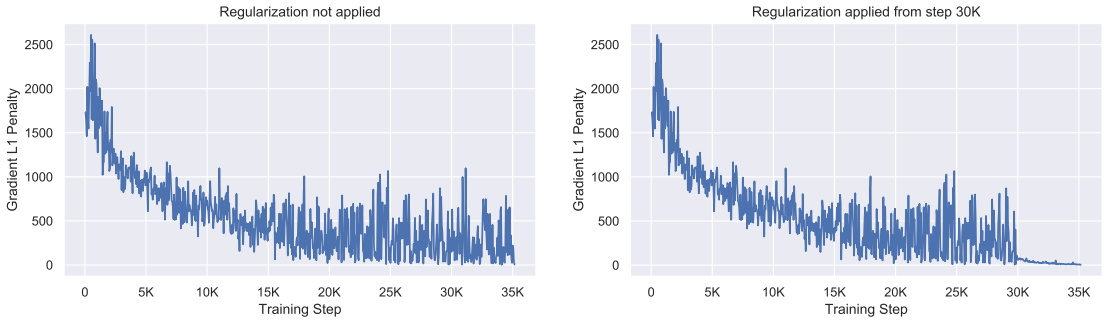
i.e., the Frobenius norm is equal to sum of the squared singular values of $\boldsymbol{W}$. From this we can conclude that the regularization term $\|\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{I}\|_2^2$ introduced by Lin et al. (2019) thus equals

$$\|\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{I}\|_2^2 = \sum_i \sigma_i^2(\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{I}) = \sum_i \left|\sigma_i^2(\boldsymbol{W}) - 1\right|^2,$$

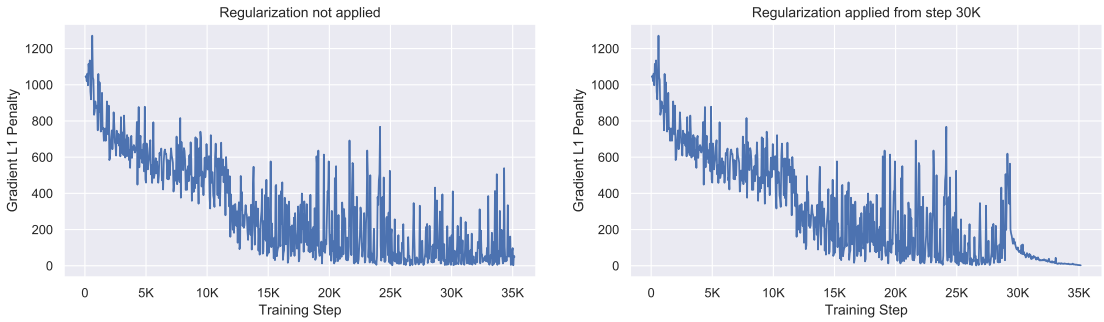
and therefore imposes a 4th power regularization term on the singular values of $\boldsymbol{W}$. A softer regularization can be introduced by regularizing $\mathrm{Tr}(\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{I})$ instead.

B.4 Gradient-Penalty Progression in Non-regularized Networks

Optimizing our regularization penalty requires computing gradients of the gradients. While this is easily done by double-backpropagation in modern software frameworks it introduces overhead (as discussed in Section 4.3.1) and makes training slower. However, as the training progresses, the gradients in unregularized networks tend to become smaller as well, which is inline with our regularization objective. It is therefore not necessary to apply the regularization from the beginning of training. In Figure B.2 we show examples of how the regularization objective naturally decreases during training. We also show how turning the regularization on in the final epochs where the regularization objective is oscillating can push the loss further down towards zero.



(a) VGG-like



(b) ResNet-18

Figure B.2: The gradients in unregularized networks tend to become smaller as training progresses. This means for large parts of the training there is no need to apply the regularization. The plots on the left show the regularization penalty in unregularized networks. The plots on the right show how turning on the regularization in the last 15 epochs of the training can push the regularization loss even further down.

B.5 ℓ_∞-bounded Perturbations and Other Norms

Figure B.3 show that the ℓ_∞-bounded perturbations include all other bounded-norm perturbations.

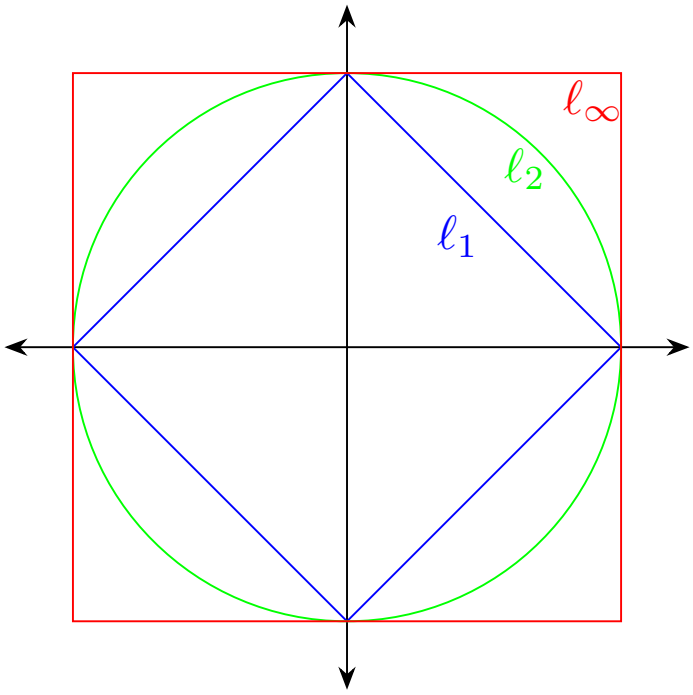


Figure B.3: ℓ_∞-bounded vectors include other bounded- norm vectors. In this plot we show that the perturbations with bounded ℓ_p-norm are a subset of ℓ_∞-bounded perturbations. For $p = 1, 2, \infty$, we plot the vectors with $\|\boldsymbol{x}\|_p = 1$.

Appendix C

Appendix to Chapter 5

C.1 Dataset Details

C.1.1 Vimeo90k

We use the Vimeo90k triplet dataset (Xue et al., 2019) containing 73,171 3-frame sequences from videos at a resolution of 448×256. We processed the dataset following Bégaint et al. (2020). The resulting dataset contains 153,939 training images and 11,346 test images.

C.1.2 FastMRI

To generate the dataset, we use the validation split from the FastMRI brain multicoil database (Zbontar et al., 2018). This contains 1378 fully sampled brain MRI images obtained through a variety of sources - T1, T1 post-contrast, T2 and FLAIR images. We then filter the dataset to only use scans from the T2 source. In addition, as the vast majority of volumes have 16 slices, we also filter by volumes with 16 slices. We then randomly split the filtered scans into a 565 training volumes and 212 testing volumes. The train dataset contains the following shapes (with their counts):

(16, 384, 384): 329

(16, 320, 320): 229

(16, 384, 312): 2

(16, 320, 260): 2

(16, 320, 240): 1

(16, 384, 342): 1

(16, 320, 270): 1

While the test dataset contains the following shapes (with their counts):

(16, 384, 384): 124

(16, 320, 320): 86

(16, 320, 260): 2

We also normalize the data to lie in [0, 1] (while COIN++ can handle data in any range, we cannot apply the image compression baselines if the data is not in [0, 1]). As the data contains outliers, we first compute a histogram of the data distribution and choose the maximum value such that 99.99% of the data has value less than this. We then normalize by the minimum and maximum value and clip any value lying outside this range (<0.01% of the data).

Disclaimer required when using the FastMRI dataset: *"Data used in the preparation of this article were obtained from the NYU fastMRI Initiative database (fastmri.med.nyu.edu) (Zbontar et al., 2018; Knoll et al., 2020). As such, NYU fastMRI investigators provided data but did not participate in analysis or writing of this report. A listing of NYU fastMRI investigators, subject to updates, can be found at:fastmri.med.nyu.edu. The primary goal of fastMRI is to test whether machine learning can aid in the reconstruction of medical images."*

C.1.3 ERA5

The climate dataset was extracted from the ERA5 database (Hersbach et al., 2019), using the processing and splits from Dupont et al. (2021b) (see this reference for details). The resulting dataset contains 12,096 grids of size 46×90, with 8510 training examples, 1166 validation examples and 2420 test examples.

C.1.4 LibriSpeech

The LibriSpeech dataset (Panayotov et al., 2015) contains several hours of read English Speech recorded at 16kHz. For training, we use the train-clean-100 split containing 28,539 examples and the test-clean split containing 2,620 examples. We train and evaluate on the first 3 seconds of every example, corresponding to 48,000 audio samples per example.

C.2 COIN Experimental Details

All models were trained using Adam for 50k iterations. We used MLPs with 2 input dimensions (corresponding to (x, y) coordinates) and 3 output dimensions (corresponding to RGB values). The coordinates were normalized to lie in $[-1, 1]$ and the RGB values were normalized to lie in $[0, 1]$. We used sine non-linearities at every layer except the last and used the initialization described in (Sitzmann et al., 2020b). We used a learning rate of 2e-4. Below we describe the architectures for each bpp level.

- 0.07bpp. Number of layers: 5, width of layers: 20.
- 0.15bpp. Number of layers: 5, width of layers: 30.
- 0.3bpp. Number of layers: 10, width of layers: 28.
- 0.6bpp. Number of layers: 10, width of layers: 40.
- 1.2bpp. Number of layers: 13, width of layers: 49.

The code to reproduce all experiments in the paper can be found at https://github.com/EmilienDupont/coin.

C.3 COIN++ Experimental Details

C.3.1 CIFAR10

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 64. All models were trained for 500 epochs (400k iterations). We used the following architectures:

- latent dim: 128, 10 layers of width 512
- latent dim: 256, 10 layers of width 512
- latent dim: 384, 10 layers of width 512
- latent dim: 512, 15 layers of width 512
- latent dim: 768, 15 layers of width 512
- latent dim: 1024, 15 layers of width 512

We used 10 inner steps at test time for all models.

COIN baseline. We manually searched for the best architecture for each bpp level. We followed all other hyperparameters from COIN (Dupont et al., 2021a) and trained for 10k iterations (we found this was enough to converge on CIFAR10). Surprisingly, we found that for CIFAR10 depth did not improve performance and that increasing the width of the layers was better. This may be because the layers are already very small.

- bpp: 3.6, 2 layers of width 12
- bpp: 4.6, 2 layers of width 14
- bpp: 5.8, 2 layers of width 16
- bpp: 7.1, 2 layers of width 18
- bpp: 8.5, 2 layers of width 20
- bpp: 10.0, 2 layers of width 22

For the COIN quantization experiments, we used uniform quantization for the weights and biases separately. We chose the number of standard deviations k at which to define the quantization range using the formula $k = 3 + 3\frac{\text{number of bits}-1}{15}$. I.e. when using 1 bit, we use 3 standard deviations and when using 16 bits we use 6 standard deviations. Indeed, there is a tradeoff between how much data we are cutting off and how finely we can quantize the range. We found that this formula generally gave robust results across different bit values.

Autoencoder baselines. All autoencoder baselines were trained using the CompressAI implementations (Bégaint et al., 2020). In order for these models to

handle 32×32 images from the CIFAR10 dataset, we modified the architectures both for BMS and CST. Specifically, for BMS we changed the last two convolutional layers in the encoder from kernel size 5, stride 2 convolutions to kernel size 3 stride 1 convolutions, in order to preserve the spatial size (we made similar changes for the transposed convolutions in the decoder). For CST we replaced the first three residual blocks in the image encoder with stride 1 convolutions instead of stride 2, hence preserving the size of the image. Similarly, we replaced the upsampling operations in the decoder with stride 1 upsampling (i.e. dimension preserving convolutions) instead of stride 2. Otherwise, we used the default parameters provided by CompressAI, i.e. for BMS, we used N=128 and M=192 and for CST N=128. We trained all models for 500 epochs with a learning rate of 1e-4. We trained models for each of the following λ values: [0.0016, 0.0032, 0.0075, 0.015, 0.03, 0.05, 0.1, 0.15, 0.3, 0.5]. As particularly CST could be unstable to train, we trained two models for each value of λ and kept the best model for the rate distortion plot.

Standard image codec baselines. We use three image codec baselines: JPEG (Wallace, 1992), JPEG2000 (Skodras et al., 2001) and BPG (Bellard, 2014). For each of these, we perform a search over either the quality, quantization level or compression ratio to find the best quality image (in terms of PSNR) at a given bpp level.

We use the JPEG implementation from Pillow version 8.1.0. We use the OpenJPEG version 2.4.0 implementation of JPEG2000, calling the binary file with

`opj_compress -i <in filepath> -r <compression ratio> -o <out filepath>`.

We use BPG version 0.9.8, calling the binary file with

`bpgenc -f 444 -q <quantization level> -o <out filepath> <in filepath>`.

Encoding time. We measure the encoding time of COIN and COIN++ on a 1080Ti GPU. For COIN we fit a separate neural network for each image in the CIFAR10 test set and report the average encoding time. For COIN++ we similarly fit modulations for each image in the test set and report the average encoding time. For BPG, we measured encoding time on an AMD Ryzen 5 3600 (12) at 3.600GHz with 32GB of RAM.

C.3.2 Kodak and Vimeo90k

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 1e-6 and batch size 64. All models were trained for 600 epochs (1.4 million iterations). We used the following architectures:

- latent dim: 16, 10 layers of width 512
- latent dim: 32, 10 layers of width 512
- latent dim: 64, 10 layers of width 512
- latent dim: 96, 10 layers of width 512
- latent dim: 128, 10 layers of width 512

We used 32×32 patches from the Vimeo90k dataset to train the model and evaluated on the full Kodak images. We used 3 inner steps for the latent dim 32 and 64 models and 10 inner steps for the latent dim 16, 96 and 128 models as this gave the best results. We quantized all modulations to 5 bits.

C.3.3 FastMRI

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 16. All models were trained for 32,000 epochs (1.1 million iterations). We used the following architectures:

- latent dim: 16, 10 layers of width 512
- latent dim: 32, 10 layers of width 512
- latent dim: 64, 10 layers of width 512
- latent dim: 128, 10 layers of width 512

We trained on 16×16×16 patches and evaluated on the full volumes. We used 10 inner steps at encoding time as this gave the best results. On the rate distortion plot, the first two points are the latent dim 16 model, quantized to 5 and 6 bits, then the latent dim 32 model, quantized to 5 bits, then the latent dim 64 model quantized to 6 bits and finally the latent dim 128 model, quantized to 5 bits and 6 bits.

C.3.4 ERA5

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 32. All models were trained for 800 epochs (210k iterations). We used the following architectures:

- latent dim: 4, 10 layers of width 384
- latent dim: 8, 10 layers of width 384
- latent dim: 12, 10 layers of width 384

We used 3 inner steps at encoding time as this gave the best results. On the rate distortion plot, the first two points are the latent dim 4 and 8 models quantized to 5 bits, then the latent dim 8 model quantized to 6 and 7 bits and finally the latent dim 12 model quantized to 7 and 8 bits.

C.3.5 LibriSpeech

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 1e-6 and batch size 64. We further scaled the coordinates to lie in $[-5, 5]$ as we found this improved performance (similar observations were made by Sitzmann et al. (2020b)). All models were trained for 1000 epochs (445k iterations), except the latent dim 256 model which was trained for 2000 epochs (890k iterations). We used the following architectures:

- latent dim: 128, 10 layers of width 512, patch size 1600
- latent dim: 128, 10 layers of width 512, patch size 800
- latent dim: 128, 10 layers of width 512, patch size 400
- latent dim: 128, 10 layers of width 512, patch size 200
- latent dim: 256, 10 layers of width 512, patch size 200

We used 3 inner steps at encoding time. On the rate distortion plot, each point corresponds to one of the above models quantized to 5, 6, 6, 7 and 7 bits respectively.

Audio codec baselines. We use the MP3 implementation from LAME version 3.100, calling the binary file with

`lame -b <bit rate> <in filepath> <out filepath>`.

C.4 COIN Additional Results

In Figure C.1, we plot the performance of COIN and JPEG at 0.3bpp (since COIN and JPEG perform similarly at this bit-rate) for all images in the Kodak dataset. As can be seen, the distortion values closely follow each other: images that are difficult for COIN to encode are also difficult for JPEG to encode.

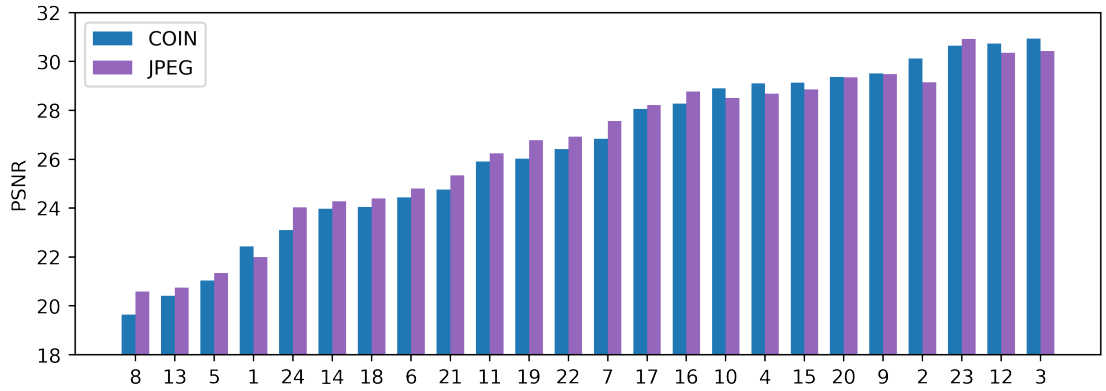


Figure C.1: Histogram of PSNR for all images in the Kodak dataset for COIN and JPEG.

C.5 COIN Qualitative Results

In Figures C.2, C.3, C.4, C.5, C.6, C.7, C.8, and C.9, we include qualitative results comparing the compression artifacts from COIN and JPEG on the Kodak dataset.

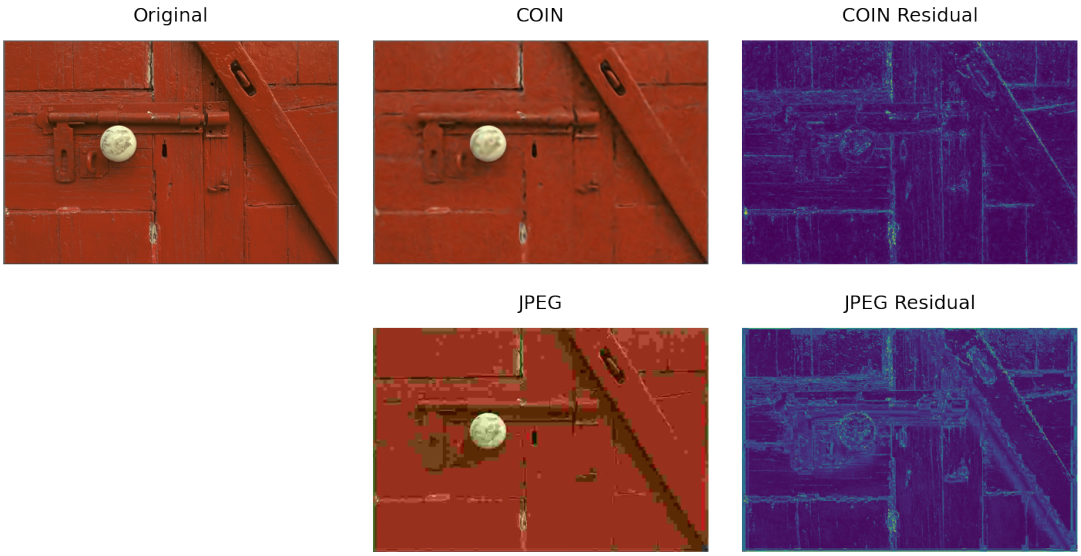


Figure C.2: Comparison between COIN and JPEG on image 2 at 0.15bpp. The PSNRs are 28.69dB and 24.67dB respectively.

Figure C.3: Comparison between COIN and JPEG on image 3 at 0.15bpp. The PSNRs are 29.02dB and 23.63dB respectively.

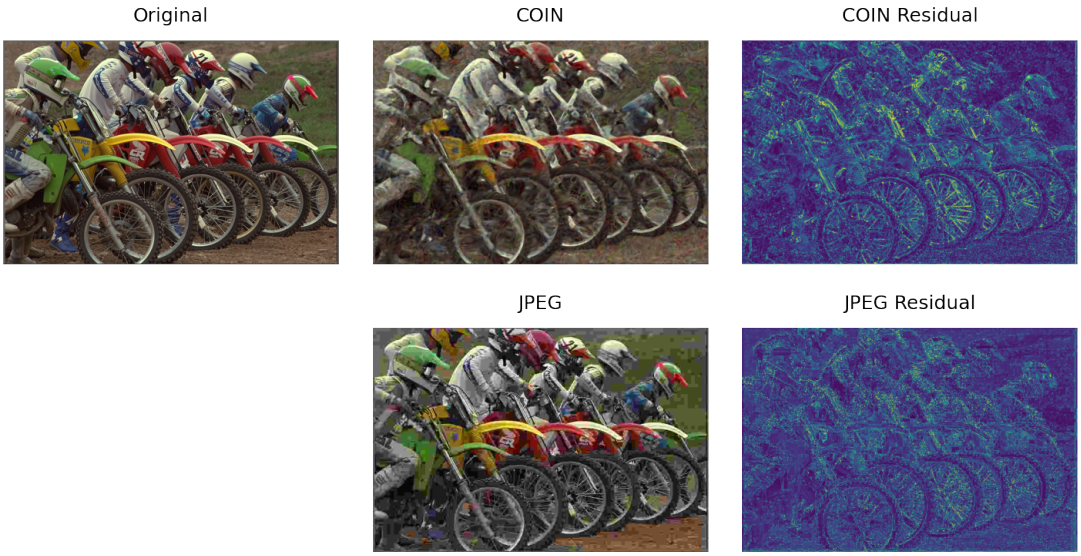


Figure C.4: Comparison between COIN and JPEG on image 5 at 0.3bpp. The PSNRs are 20.97dB and 21.34dB respectively.

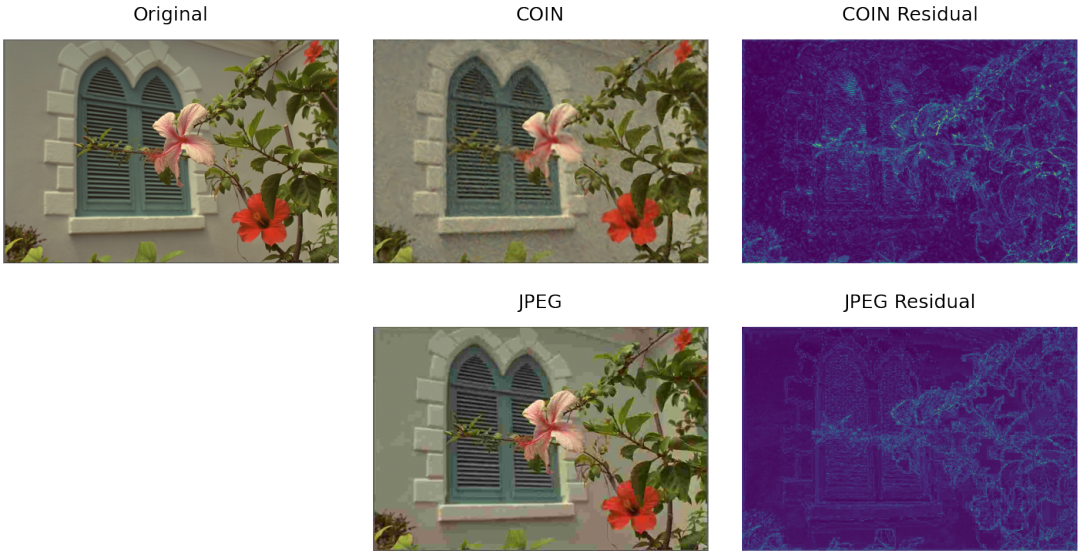


Figure C.5: Comparison between COIN and JPEG on image 7 at 0.3bpp. The PSNRs are 26.92dB and 27.56dB respectively.

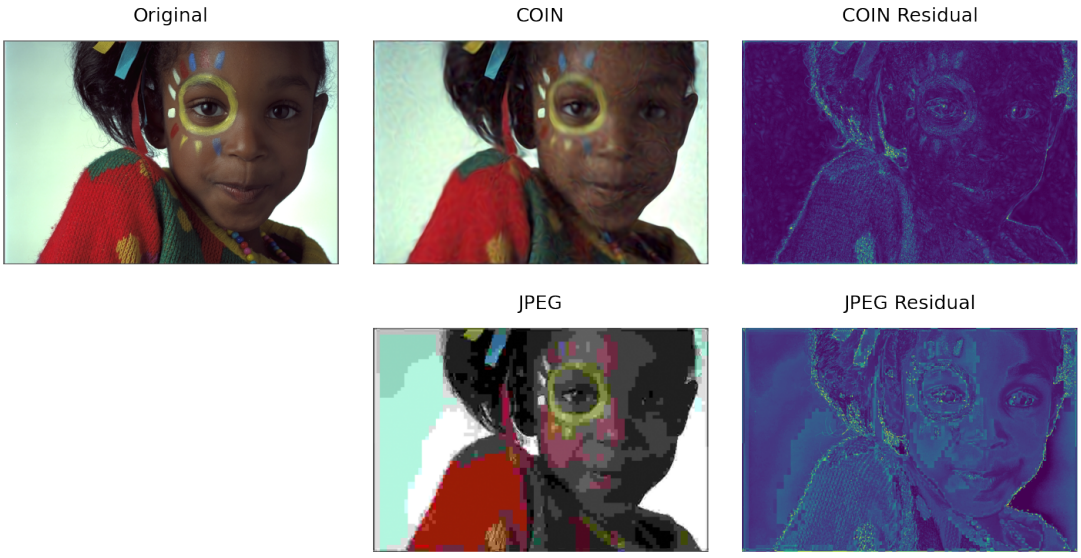


Figure C.6: Comparison between COIN and JPEG on image 15 at 0.15bpp. The PSNRs are 27.35dB and 21.74dB respectively.

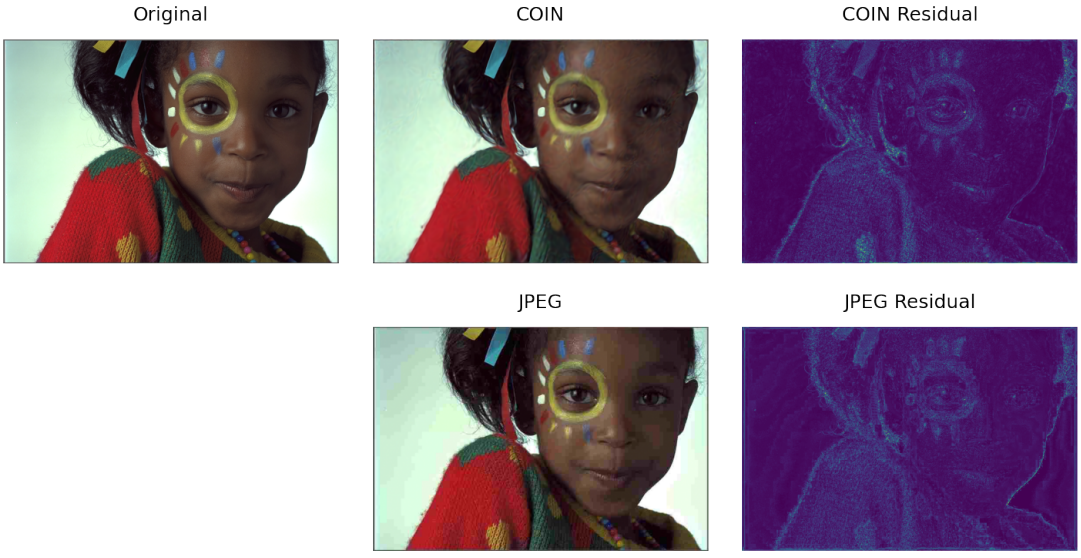


Figure C.7: Comparison between COIN and JPEG on image 15 at 0.3bpp. The PSNRs are 29.31dB and 28.85dB respectively.

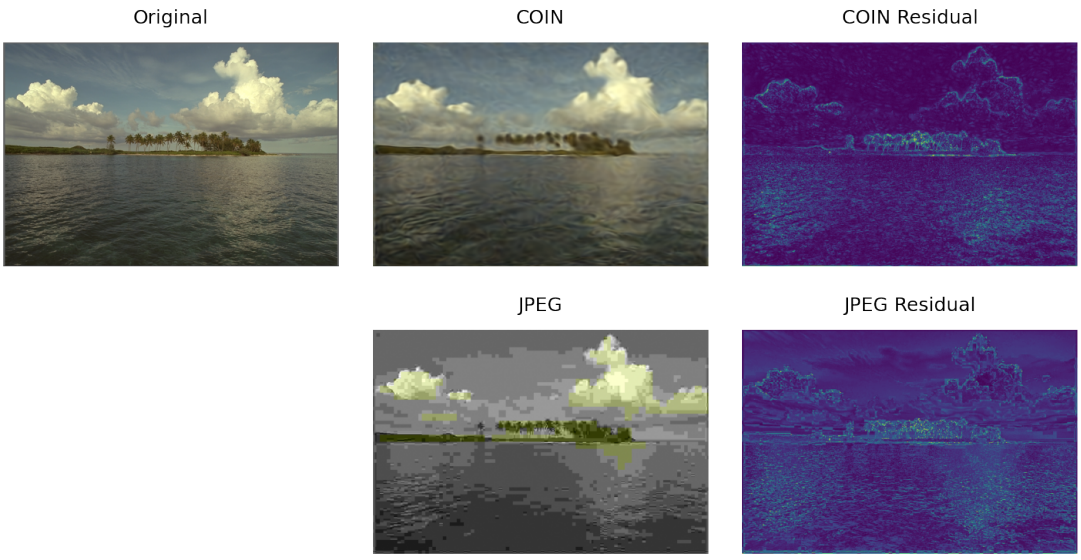


Figure C.8: Comparison between COIN and JPEG on image 16 at 0.15bpp. The PSNRs are 27.19dB and 24.16dB respectively.

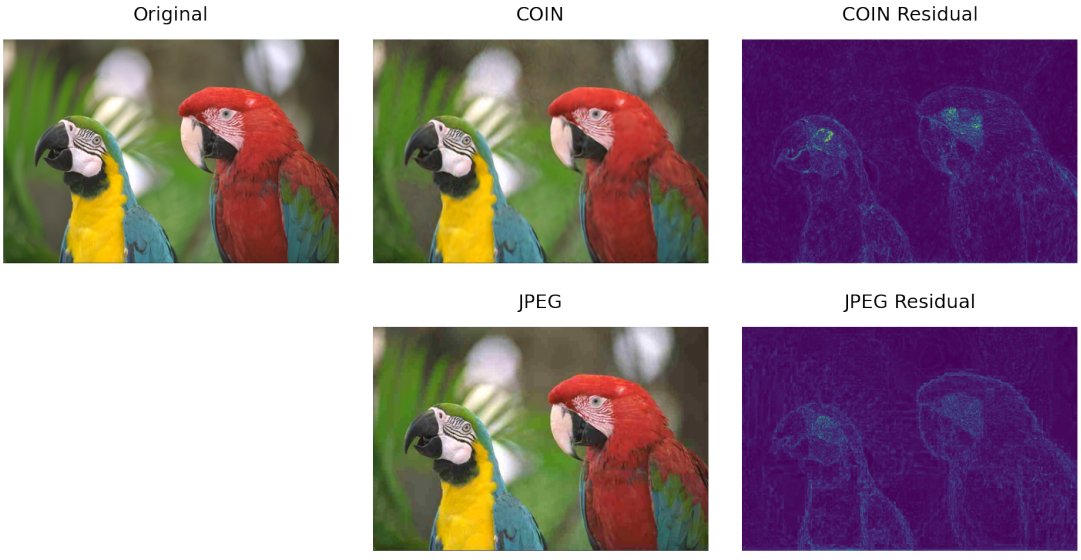


Figure C.9: Comparison between COIN and JPEG on image 23 at 0.3bpp. The PSNRs are 31.08dB and 30.92dB respectively.

C.6 Figure Details

Figure 5.10b (COIN vs COIN++ quantization). The results in this figure are averaged across the entire CIFAR10 test set. We used COIN and COIN++ models that achieve roughly the same PSNR (30-31dB), corresponding to the bpp 7.1 model for COIN and the latent dim 384 model for COIN++.

Figure 5.10d (Encoding time). The BPG model uses 1.25 bpp (PSNR: 28.7dB), the COIN++ model 1.14 bpp (PSNR: 28.9dB) and the COIN model 7.1 bpp (PSNR: 30.7dB).

Figure 5.12 (Kodak qualitative samples). The COIN++ model used for this plot has a bpp of 0.537 (latent dim 128).

Figure 5.14 (FastMRI qualitative samples). The COIN++ model used for this plot has a bpp of 0.168 (latent dim 128).

Figure 5.11 (ERA5 qualitative samples). The COIN++ model used for this plot has a bpp of 0.012 (latent dim 8).

Figure C.13 (Qualitative quantization). This figure uses the COIN++ model with a latent dim of 768.

C.7 Failed Attempts

- As MAML is very memory intensive, we experimented with first-order approximations. We ran first-order MAML as described in Finn et al. (2017b), but found that this severely hindered performance. Further, methods such as REPTILE (Nichol et al., 2018) are not applicable to our problem, as the weights updated in the inner and outer loop are not the same.
- Mehta et al. (2021) use a similar approach of overlapping patches. However, we found that using overlapping patches yielded a worse tradeoff between reconstruction accuracy and number of modulations and therefore used non-overlapping patches throughout.
- We experimented with using a deep MLP (and the architecture from Mehta et al. (2021)) for mapping the latent vector to modulations but found that

this decreased performance. As MLPs are strictly more expressive than linear mappings, we hypothesize that this is due to optimization issues arising from the meta-learning. If the base network is learned without meta-learning, it is likely a deep MLP would improve performance over a linear mapping.

C.8 Meta-learning Curves

Figure C.10 shows the meta-learning training curves of COIN++. Meta-training is occasionally be unstable, although the model typically recovers from loss instabilities.

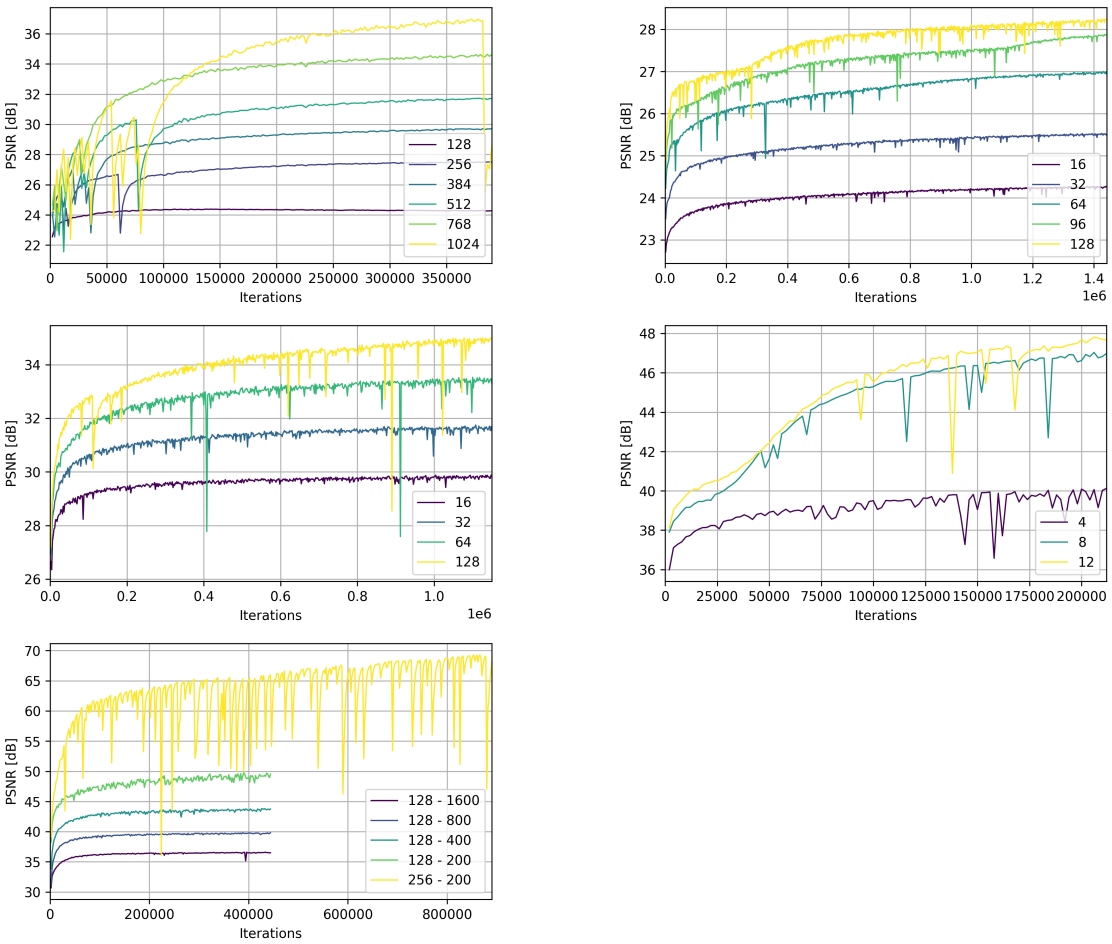


Figure C.10: Validation PSNR (3 inner steps) for meta-learning on CIFAR10 (top left), Kodak (top right), FastMRI (middle left), ERA5 (middle right) and LibriSpeech (bottom). Note that for LibriSpeech, the legend corresponds to "latent dimension - patch size".

C.9 CIFAR10 Ablations

Figure C.11 shows rate distortion plots for full precision, quantized and entropy coded modulations.

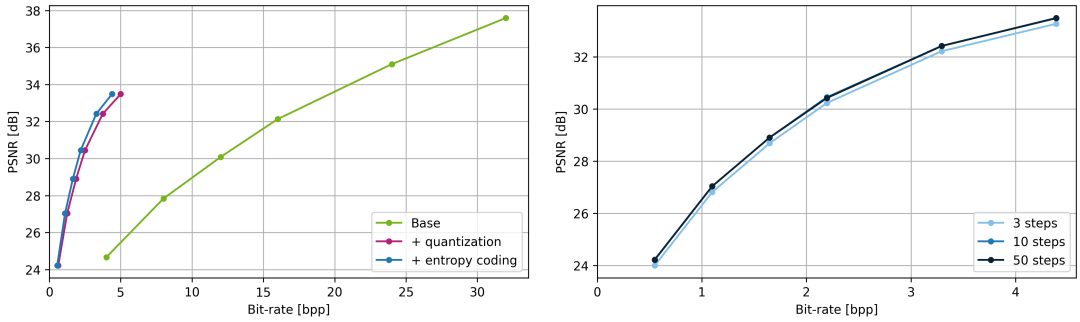


Figure C.11: (Left) Effect of of quantization (to 5 bits) and entropy coding on CIFAR10. (Right) Effect of number of inner steps on CIFAR10 for a model that has been quantized to 5 bits, with entropy coding. While we use 3 inner steps for meta-learning, performing 10 steps at test time leads to an increase in reconstruction performance of 0.5-1.5dB, while fitting for more than 10 steps generally does not improve performance. Indeed, curves for 10 and 50 steps almost fully overlap.

C.10 COIN++ Encoding Curves

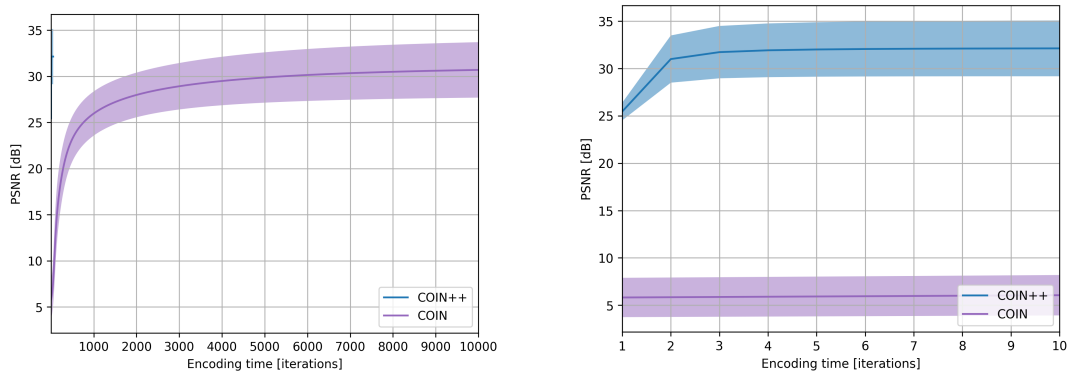


Figure C.12: Encoding curves for COIN and COIN++ on CIFAR10 (full curve on the left, zoomed in version on the right). The COIN model has a bpp of 7.1, while COIN++ has a bpp of 2.2.

C.11 Qualitative Quantization Results

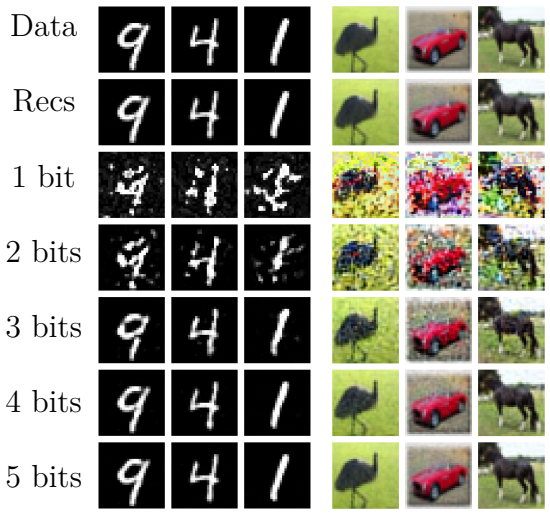


Figure C.13: Qualitative effects of quantization. The top row shows ground truth data from MNIST and CIFAR10, the second row shows the reconstructions from full precision (32 bit) modulations. The subsequent rows show reconstructions when quantizing to various bitwidths. As can be seen, with only 5 bits, reconstructions are nearly perfect. Using as few as 1 or 2 bits, the class of the object is generally recognizable.

C.12 Additional Qualitative Results

Figures C.14, C.15, and C.16 include addition COIN++ qualitative results on ERA5, Kodak, and FastMRI datasets.

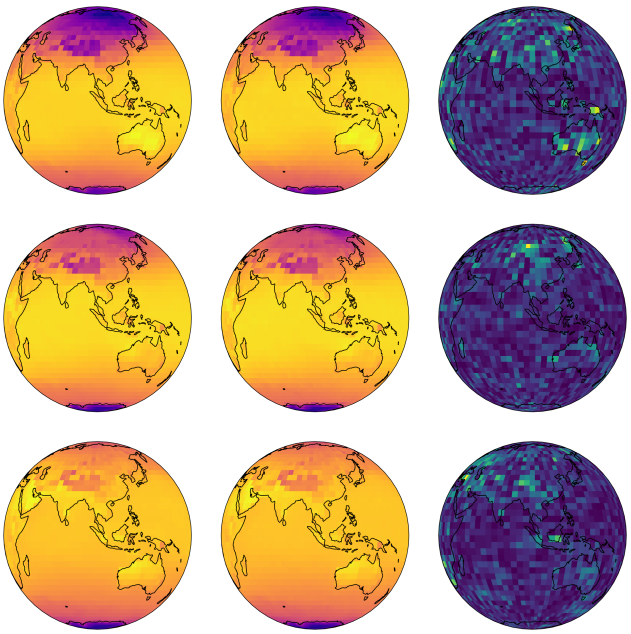


Figure C.14: Qualitative compression artifacts on ERA5 using latent dim 8, and .012bpp model (original in the first column, COIN++ in second, and residual in the third).

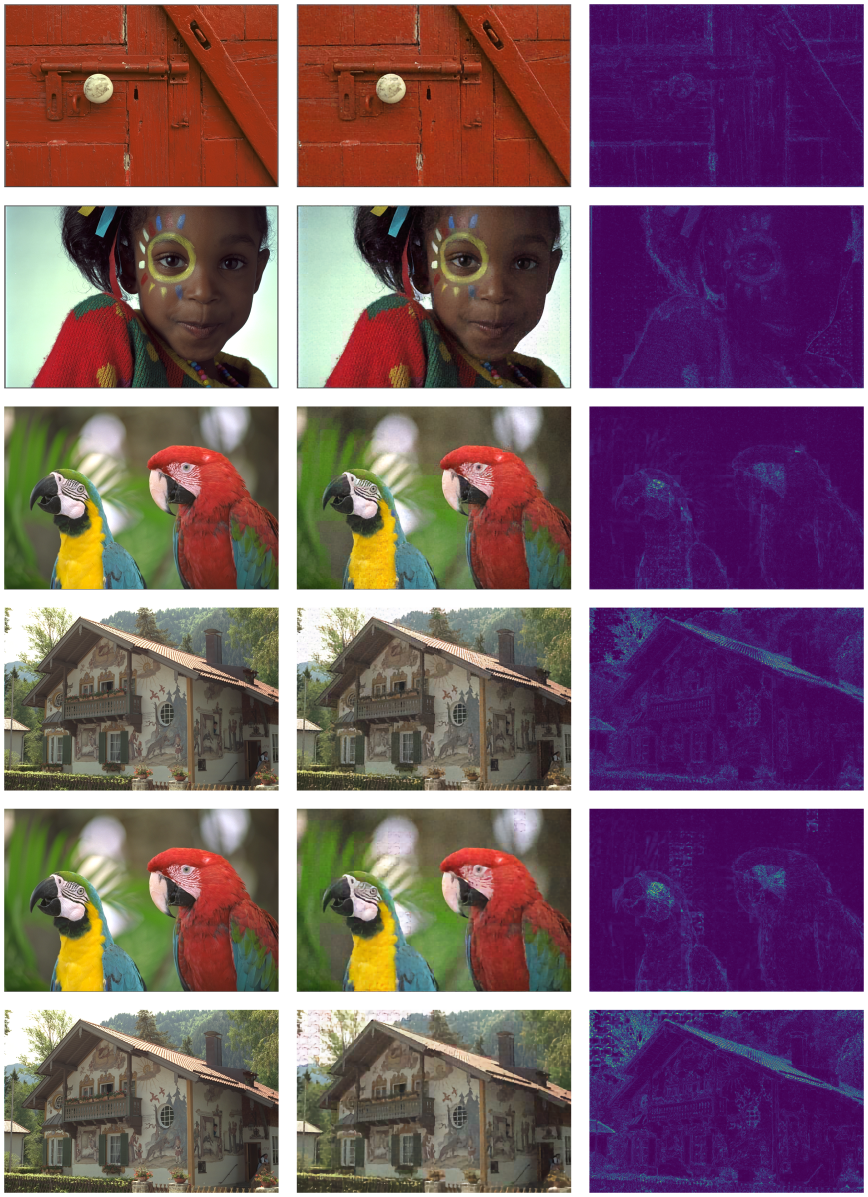


Figure C.15: Qualitative compression artifacts on Kodak (original in first column, COIN++ in second column and residual in third column). The first 4 rows correspond to the model with latent dim 128 (0.537 bpp), while the bottom two rows correspond to the model with latent dim 64 (0.398 bpp).

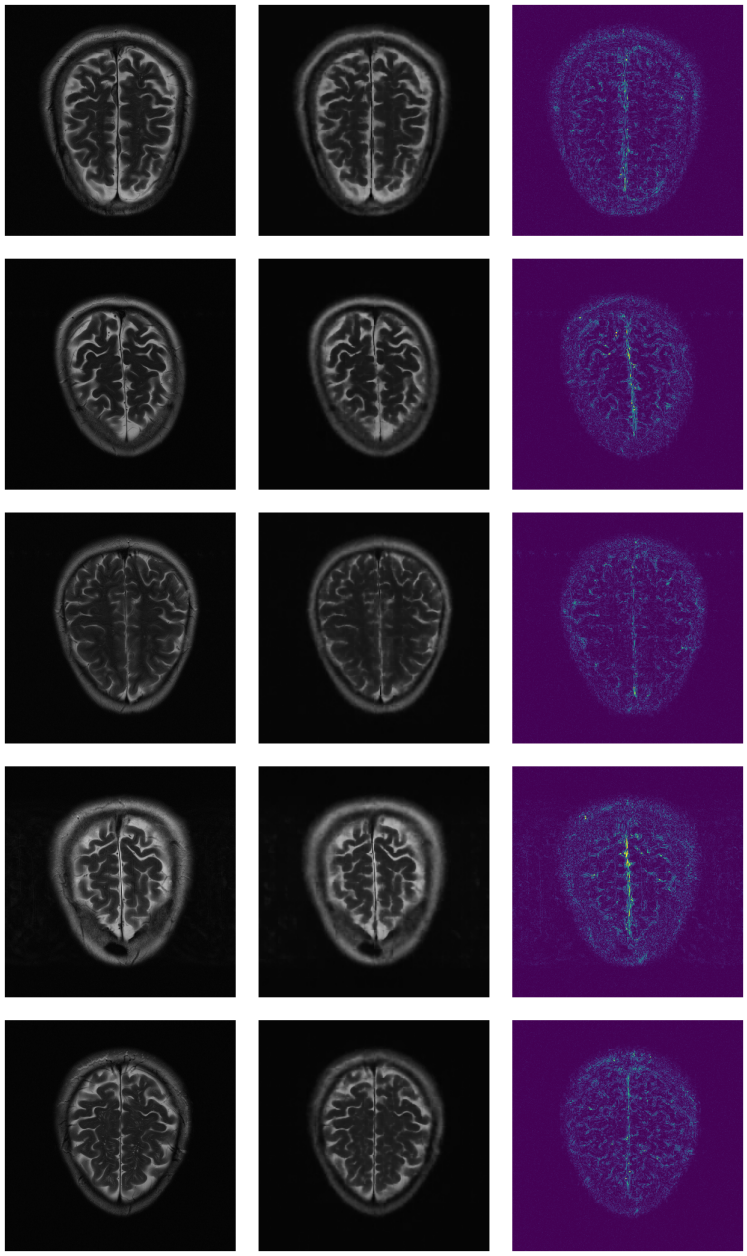


Figure C.16: Qualitative compression artifacts on FastMRI using the latent dim 128 model with 0.168 bpp (original in first column, COIN++ in second column and residual in third column).