CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

The Impact of Deep Neural Network Pruning on the Hyperparameter Performance Space: An Empirical Study

Master's Thesis in Computer Science and Engineering

JONNA MATTHIESEN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

The Impact of Deep Neural Network Pruning on the Hyperparameter Performance Space: An Empirical Study

Jonna Matthiesen

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Understanding Hyperparameters under Deep Neural Network Pruning

Jonna Matthiesen

Supervisor: Yinan Yu, Department of Computer Science and Engineering
Advisors: Andreas Ask, Deep Learning Researcher, Embedl &
Daniel Ödman, Deep Learning Researcher, Embedl
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Understanding Hyperparameters under Deep Neural Network Pruning

Jonna Matthiesen
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

With the continued growth of deep learning models in terms of size and computational requirements, the need for efficient models for deployment on resource-constrained devices becomes crucial. Structured pruning has emerged as a proven method to speed up models and reduce computational requirements. Structured pruning involves removing filters, channels, or groups of operations from a network, effectively modifying its architecture. Since the optimal hyperparameters of a model are tightly coupled to its architecture, it is unclear how pruning affects the choice of hyperparameters. To answer this question, we investigate the impact of deep neural network pruning on the hyperparameter performance space.

In this work, we perform a series of experiments on popular classification models, ResNet-56, MobileNetV2, and ResNet-50, using CIFAR-10 and ImageNet datasets. We examine the effect of uniform and non-uniform structured magnitude pruning on the learning rate and weight decay. Specifically, we explore how pruning affects their relationship and the risk associated with not tuning these hyperparameters after pruning. The experiments reveal that pruning does not have a significant impact on the learning rate and weight decay, suggesting that extensive hyperparameter tuning after pruning may not be crucial for optimal performance.

Overall, this study provides insights into the complex dynamics between pruning, model performance, and optimal hyperparameters. The findings give guidance for optimising and fine-tuning pruned models and contribute to advancing model compression and hyperparameter tuning, highlighting the interplay between model architecture and hyperparameters.

Keywords: Compression, Deep Learning, DNN, Hyperparameters, Optimization, Pruning, Hyperparameter Optimisation, Hyperparameter Tuning

Acknowledgements

I would like to thank Yinan Yu for her tremendous support throughout this bumpy journey. Yinan has gone above and beyond in ensuring the success of my master's thesis, not only in terms of research but also in navigating the bureaucratic aspects and hurdles. Thank you! Further, I would also like to thank my advisors at Embedl, Andreas Ask and Daniel Ödman, for their support and guidance throughout this work. The combined efforts from Yinan, Daniel, and Andreas have been enriching and have played a crucial role in shaping the quality of this work.

Jonna Matthiesen, Gothenburg, 2023-06-21

Contents

List of Figures

List of Tables

List of Acronyms

CNN	Convolutional Neural Network
DNN	Deep Neural Network
FLOP	Floating Point Operation
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
HPO	Hyperparameter Optimisation
LR	Learning Rate
NAS	Neural Architecture Search
NLP	Natural Language Processing
NN	Neural Network
ROI	Region of Interest
SDK	Software Development Kit
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

In the past couple of years, rapid advancements in the field of deep learning have led to the development of a variety of applications across multiple fields, such as computer vision (Dosovitskiy et al., 2021; K. He et al., 2015b), natural language processing (Brown et al., 2020; Devlin et al., 2019), and generative modelling (Child, 2021; OpenAI, 2023). Despite these promising advancements, there is a cost to pay: the increasing network size. Deep neural networks (DNNs) are becoming deeper and wider, requiring more parameters and computational resources for training and evaluation. The increasing network size poses significant challenges as larger models with more parameters require more computational resources resulting in increased inference latency, memory footprint, and energy consumption. Consequently, deploying them on resource-constrained devices becomes challenging (Fan et al., 2021).

To address these challenges, the field of deep neural network compression has emerged. Research in this field aims to reduce the size and computational requirements of deep learning models while minimising the impact on model performance. One approach to model compression is deep neural network pruning. This technique involves removing unnecessary weights or structures from a model, resulting in improved efficiency without significant performance compromise.

Pruning has become an integral part of deploying DNNs on resource-constrained devices, such as edge devices, embedded systems, and smartphones. By adopting the right pruning strategies given the constraints of each individual hardware, it is possible to significantly reduce a network's size and latency without considerably affecting its performance (Choudhary et al., 2020; Liu et al., 2019; Sui et al., 2023).

Pruning is a versatile technique applicable at different stages of the training process, including during, after, and even before training, as shown in Figure 1.1. Thus, pruning is tightly coupled to the act of training. A key element in training DNNs is the choice of hyperparameters. In today's context, where modern models often require significant training time ranging from days to months, efficient and effective methods for identifying the optimal set of hyperparameters are more important than ever. Despite extensive research in this area, setting hyperparameters is still heavily based on intuition, requiring years of experience to acquire (L. N. Smith, 2018).

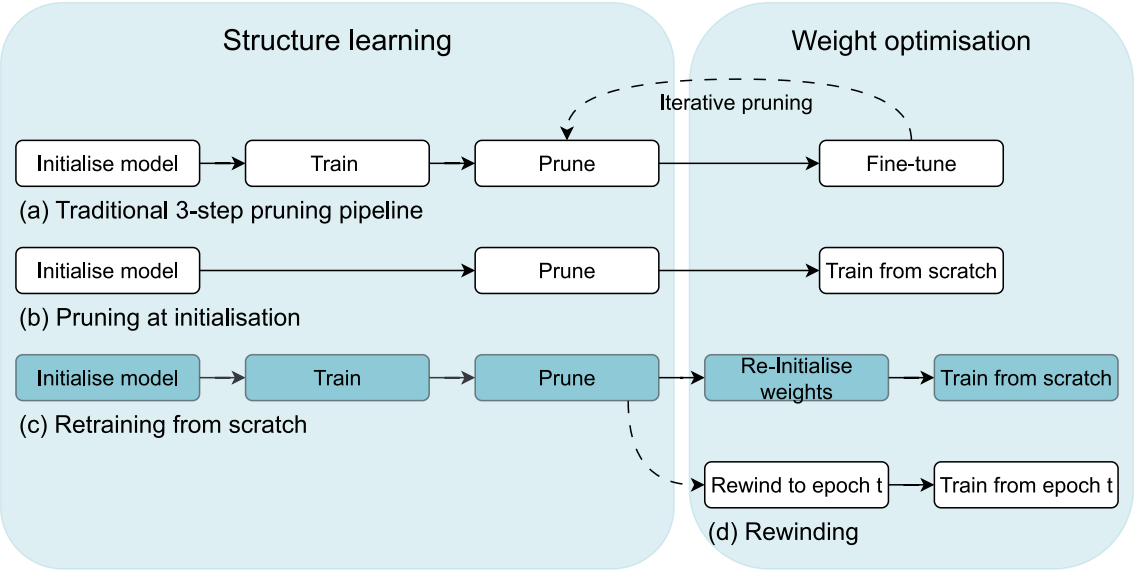

Figure 1.1: Overview of neural network pruning pipelines in the literature. The pipelines can roughly be divided into two phases: structure learning, where the network's structure, including its weights, is learned or randomly set as in (b); and weight optimisation, where the lost accuracy after pruning is regained. Rewinding (d) involves resetting, for example, hyperparameters or model weights to their state at epoch t. The highlighted pipeline (c) indicates the approach used in this work. Adopted from (Y. Wang et al., 2019).

Structured pruning, where entire filters, channels or groups of operations are removed, effectively changes the architecture of the network and allows for reduced network size and improved latency on hardware (Sui et al., 2023). Since the optimal hyperparameters of a model are tightly coupled to its architecture, it is unclear how pruning affects the choice of hyperparameters. Thus, in this study, we investigate the impact of structured pruning on the hyperparameter performance space of a model, aiding the development of effective and efficient pruning techniques by addressing the challenge of hyperparameter optimisation. To achieve this, we focus on two primary research questions:

Q1 *What are the effects of structured pruning on the hyperparameter performance space of a model?*

Q2 *What is the performance risk when not tuning hyperparameters after pruning?*

By addressing these research questions, we aim to explore the complex interplay between structured pruning, hyperparameters, and model performance. In this empirical study, we utilise state-of-the-art classification models on popular benchmark datasets in computer vision, the CIFAR-10 and ImageNet datasets.

To our best knowledge, we are the first to investigate the impact of structured pruning on the optimal hyperparameters of a model. This study contributes to a deeper understanding of the relationship between hyperparameters and model performance of DNNs. This knowledge can help in answering the practical question of whether time-consuming hyperparameter optimisation after pruning is necessary or if the initial optimal hyperparameters of the unpruned model are sufficient.

Chapter 2

Theory

The purpose of this chapter is to establish the theoretical foundations for the subsequent study on the impact of deep neural network pruning on the hyperparameter performance space of a model. This chapter does not aim to provide a comprehensive review of relevant topics but rather contextualises them within the framework of this study. In addition to the theoretical concepts, we review existing literature and related work and highlight the relevance of the proposed work within the current research landscape and practical applications. To this end, we look at the following areas:

2.1 Deep Learning In this section, we introduce the fundamental concepts of deep learning, focusing on its application in computer vision utilising convolutional neural networks (CNNs). Two CNN architectures are relevant to this study: ResNets and MobileNets. Additionally, this section discusses the current limitations and challenges when deploying DNNs on resource-constrained devices, emphasising the need for efficient and lightweight models.

2.2 Neural Network Pruning This section provides an overview of deep neural network pruning. We discuss various pruning techniques and explore how they address the challenges of deep learning raised in the previous section. We present a classification of pruning based on its structure (unstructured vs structured), strength (pruning ratio), and method (uniform vs non-uniform), and discuss how pruning fits into the training pipeline of a model.

2.3 Hyperparameters in Deep Learning In this section, we delve into the role of hyperparameters in deep learning and their impact on both the training process and model performance. We emphasise the importance of hyperparameter optimisation, which plays a vital role in achieving optimal results, and discuss related work. Moreover, we introduce the concept of the hyperparameter performance space, which is of fundamental importance to this study.

2.1 Deep learning

Deep learning is a subfield of machine learning which involves the training of DNNs to learn from data. A DNN typically consists of interconnected nodes organised in layers, with each layer represented by learnable parameters. These parameters, such as weights and biases, are utilised to compute the layer's output. The input layer takes raw data, while the output layer generates the final predictions.

In contrast to traditional neural networks (NNs), DNNs can have numerous hidden layers between the input and output layers, hence the term *deep*. This architecture allows DNNs to learn features at multiple levels of abstraction, enabling them to capture complex patterns and relationships in data. As a result, deep learning has achieved notable success in a wide range of domains that are considered challenging for classical machine learning techniques. Depending on the task and domain, a range of different deep learning architectures and types have evolved, ranging from simple feed-forward neural networks to more complex types like the transformer or convolutional neural networks.

The transformer architecture, for instance, revolutionised natural language processing (NLP) tasks by effectively capturing contextual relationships using self-attention mechanisms (Vaswani et al., 2017). Today, models such as the generative pre-trained transformer (GPT) dominate the field, achieving state-of-the-art performance in a wide range of NLP tasks (OpenAI, 2023).

In this study, our focus is on convolutional neural networks. While transformers were originally introduced in the field of NLP, CNNs have played a pivotal role in revolutionising the field of computer vision and have demonstrated remarkable success in image and video processing applications (K. He et al., 2015b; A. Howard et al., 2019; Tan & Le, 2020).

2.1.1 Convolutional neural networks

Convolutional neural networks are one class of deep learning architectures specifically designed to process data with a matrix-like structure, such as images. Unlike traditional neural networks that treat input data as a flat structure, CNNs take advantage of the underlying spatial relationships. The main building blocks in a CNN are convolutional and pooling layers, allowing for the extraction of local patterns while maintaining shift and distortion invariance (Lecun et al., 1989).

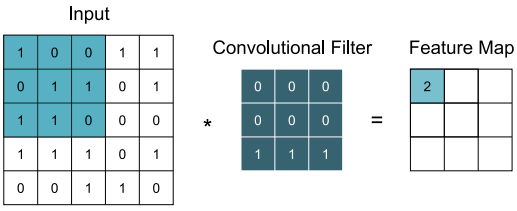

Figure 2.1: An example of a convolutional operation with a 3×3 convolutional filter with stride 1.

A convolutional layer consists of multiple filters, each of which is learned during training to capture local patterns and features such as edges, textures, and shapes from the input. These filters, also known as kernels, are small matrices of learnable

parameters and perform convolutional operations on the input. A convolutional operation is the sum of element-wise multiplications between the kernel and a small region of the input, as illustrated in Figure 2.1. The stride describes the step size at which the filter moves across the input. Although these filters are typically small in spatial dimensions, they are applied to the entire depth of the input (O'Shea & Nash, 2015).

To lower the spatial size of the feature maps generated by the convolutional layer and to introduce invariance to small changes in the input, a pooling layer is often added after the convolutional layer. The pooling layer reduces the number of parameters and the computational complexity of the model by subsampling the feature map, see Figure 2.2. Common pooling operations include Max, Average, or Min Pooling, where a small region of the input is subsampled by its local maximum, average, or minimum, respectively. These operations serve the purpose of summarising important features locally, making the model more robust to small variations like shifts and other distortions (O'Shea & Nash, 2015).

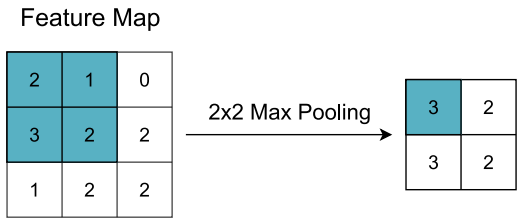

Figure 2.2: An example of a Max Pooling operation (stride = 1) performed on a feature map.

While CNNs have been the dominant models in computer vision for several decades and have achieved remarkable success in various tasks, modern models like Vision Transformers have shown themselves to be a viable alternative. For example, Dosovitskiy et al. (2021) apply transformers to computer vision tasks and achieve promising results in image classification. However, CNNs still hold several advantages, such as their widely studied interpretability (Olah et al., 2017) and their ability to leverage transfer learning from pre-trained models (Shin et al., 2016). Moreover, CNNs benefit from a well-established research community, extensive toolkits and frameworks, and a vast collection of available pre-trained models, making them a practical and reliable choice for many computer vision applications.

In this work, we focus on the application of CNNs to the task of image classification. Image classification, also referred to as image recognition, often serves as a standard benchmark for CNNs as it is a fundamental task in computer vision (Lin et al., 2015; Russakovsky et al., 2015). Given an input image, a deep learning classification model usually outputs confidence scores for the individual classes, with the highest score being the predicted class of the input image.

In the following, we look at two CNN architecture families relevant to this study: residual and mobile networks. Our aim is to provide the reader with a concise overview of their design and architectural components. This lays the foundation for understanding how pruning techniques can be effectively applied to these models.

Residual networks

One family of CNNs that has become increasingly popular in recent years are residual networks (ResNets) (K. He et al., 2015b). ResNets address the problem of vanishing

gradients, a common issue in deep learning. Vanishing gradients refer to the problem where the gradients in backpropagation become too small to effectively update earlier layers in the network, leading to slow convergence and poor performance (K. He et al., 2015b).

The ResNet architecture consists of stacked *residual blocks*, where each block includes a skip connection (K. He et al., 2015b). The residual block is composed of two or more convolutional layers, with each layer typically followed by a batch normalisation layer and a ReLU activation function, see Figure 2.3. The skip connection connects the input of the block to its output, allowing the information to bypass the convolutional layers. By doing so, the skip connection ensures that the information from earlier layers can directly influence the output of later layers, preventing the gradients from vanishing and allowing deeper feature learning (K. He et al., 2015b).

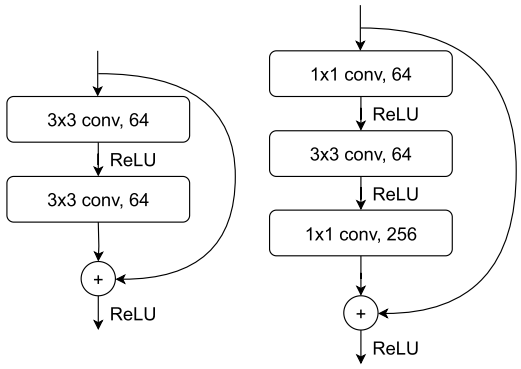

Figure 2.3: Illustration of a residual block for ResNet-56 (left) and ResNet-50 (right). The skip connection adds the input to the output of the stacked convolutional layers. Adapted from (K. He et al., 2015b).

The application of ResNets has shown remarkable performance on a wide range of computer vision tasks. In 2015, ResNet was the winning architecture of the ImageNet and Microsoft COCO Visual Recognition Challenges (Lin et al., 2015; Russakovsky et al., 2015) for the tasks of ImageNet detection and localisation, and COCO detection and segmentation. In addition to their excellent performance, ResNets can be easily scaled up or down to meet the requirements of different tasks, making them highly adaptable to different applications (K. He et al., 2015b).

Two residual networks relevant to this study are ResNet-50 and ResNet-56, see Table 2.1. ResNet-50 is a 50-layer residual network with over 25 million parameters, taking images of size $224 \times 224 \times 3$ as input. ResNet-56, a 56-layer network, takes input images of size $32 \times 32 \times 3$ and comprises about 0.85 million parameters (K. He et al., 2015b).

ResNet-50	ResNet-56
$7 \times 7, 64$, stride 2	
3×3 Max Pooling, stride 2	
$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	
$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 9$
$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 9$
$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 9$
Global Average Pooling	
fully-connected layer	

Table 2.1: Architecture of ResNet-50 and ResNet-56. If not other indicated convolutional layers have stride 1 and are described by $a \times a,\ b$ with $a, b \in \mathbb{N}$, where $a \times a$ is the size and b the number of kernels. Blocks of convolutional layers describe residual blocks, see Figure 2.3. Adapted from (K. He et al., 2015b).

Mobile networks

While ResNets are large networks designed for high-performance tasks on powerful hardware, they might not be optimal for deployment on devices with limited computational resources. MobileNets are a type of lightweight CNN designed for computer vision tasks on mobile and embedded devices (A. G. Howard et al., 2017). MobileNets have achieved excellent results on a wide range of image classification tasks while being significantly smaller and computationally more efficient than traditional deep CNNs (A. G. Howard et al., 2017). The MobileNet architecture was first introduced by A. G. Howard et al. (2017) (MobileNetV1) and has since become popular in the computer vision community, with variants such as MobileNetV2 (Sandler et al., 2019) and MobileNetV3 (A. Howard et al., 2019).

MobileNetV1 utilises *depthwise separable convolutions* that decompose a standard convolution into a depthwise convolution and a pointwise convolution, reducing the computational cost by 8 to 9 times (A. G. Howard et al., 2017). In contrast to classical convolutional layers where the filter is applied to the whole depth of the input, *depthwise convolutional layers* apply a single filter to each input channel. *Pointwise convolutions* are 1×1 convolutional layers that are applied to the output of

the depthwise convolution (A. G. Howard et al., 2017). Figure 2.4 shows a comparison of the classical convolutional operation with the depthwise separable convolutional operation. Compared to ResNet-50, MobileNetV1 is a rather small network with 4.2 million parameters, comprising only 13 stacked depthwise separable convolutional layers, totalling 28 layers when counting depthwise and pointwise convolutions as separate layers (A. G. Howard et al., 2017).

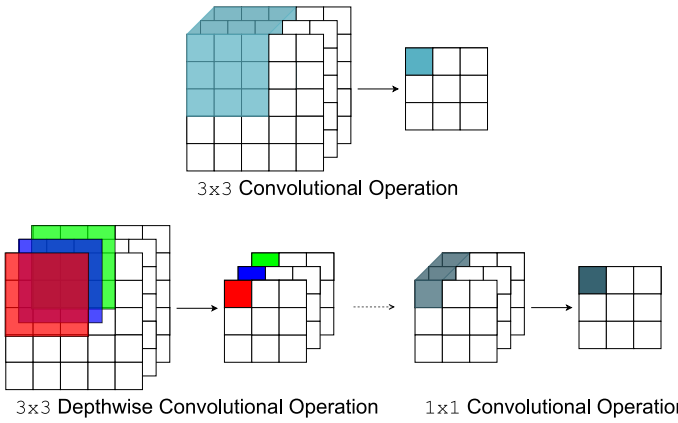

Figure 2.4: Comparison of classical convolution (top) with depthwise separable convolution (bottom). Adapted from (A. G. Howard et al., 2017).

One variant relevant to this study is MobileNetV2. Compared to MobileNetV1, MobileNetV2 achieves 1.4% better accuracy on the ImageNet dataset, a common dataset in computer vision for benchmarking (Russakovsky et al., 2015). MobileNetV2 replaces the depthwise separable convolutional block of the original MobileNetV1 architecture with inverted residual blocks, resulting in a deeper architecture with 53 layers (Sandler et al., 2019). These blocks are the main building structure of MobileNetV2 and are shown in Figure 2.5. Inverted residual blocks wrap a depthwise convolutional layer in 1×1 convolutional layers. The first 1×1 convolutional layer expands the input along its depth, increasing the number of channels; the second 1×1 convolution projects the depthwise convolutional output back to the lower-dimensional space of the input. Due to the combination of compression and expansion of the input, inverted residual blocks are often referred to as bottleneck blocks (Sandler et al., 2019).

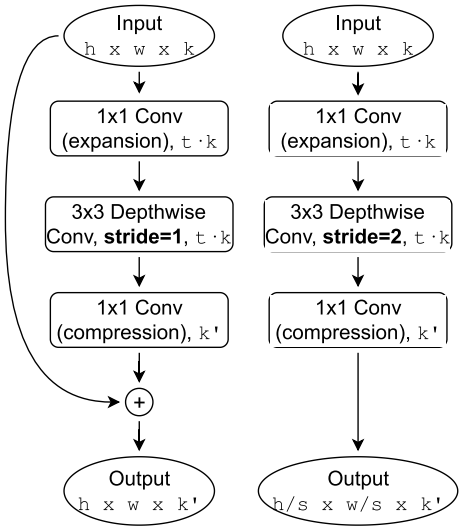

Figure 2.5: Illustration of a residual block for MobileNetV2. Inverted residual blocks transform the input of shape $h \times w \times k$ to shape $\frac{h}{s} \times \frac{w}{s} \times k'$, where s is the stride of the depthwise convolutional layer. Adapted from (Sandler et al., 2019).

MobileNetV2 was originally proposed for an input shape of $224 \times 224 \times 3$. However, in this study, we modify the architecture of MobileNetV2 for smaller images with dimensions of $32 \times 32 \times 3$. To make MobileNetV2 compatible with low-resolution inputs, it is common to disable some downsampling layers by replacing stride 2 with stride 1 in certain layers as described by Ayi and El-Sharkawy (2020). Table 2.2 displays the architecture of the original MobileNetV2 and highlights the changes for the adapted version. For the purpose of this study, we refer to this modified version of MobileNetV2 as 'MobileNetV2' when the input shape is clear from the context.

Operator	expansion factor	# channels	× n	stride
3×3 Conv layer	-	32	1	**2**
Bottleneck	1	16	1	1
Bottleneck	6	24	2	**2**
Bottleneck	6	32	3	**2**
Bottleneck	6	64	4	2
Bottleneck	6	96	3	1
Bottleneck	6	160	3	2
Bottleneck	6	320	1	1
1×1 Conv layer	-	1280	1	1
Global Average Pooling	-	-	1	-
Fully-connected layer	-	-	-	-

Table 2.2: Architecture of MobileNetV2. Each line describes a sequence of n identical operations. In each sequence, the first layer has stride s and all subsequent layers use stride 1. See Figure 2.5 for a description of the bottleneck blocks. For the modified MobileNetV2 with input size $32 \times 32 \times 3$ some downsample layers (marked in bold) are disabled, i.e. stride 2 is replaced with stride 1. Adapted from (Ayi & El-Sharkawy, 2020; Sandler et al., 2019)

2.1.2 Challenges of deploying large models

Deep learning has revolutionised the field of machine learning by enabling the development of highly accurate models for a wide range of applications. However, their effectiveness comes at a cost. DNNs become increasingly larger and more resource-intensive, presenting significant challenges (Fan et al., 2021).

Modern DNNs can have hundreds of millions or even billions of parameters (Sevilla et al., 2022). For instance, the transformer-based language model GPT-3 developed by OpenAI, contains roughly 175 billion parameters. Similarly, recent success in computer vision is tightly coupled with making models deeper and wider; modern CNNs can have tens to hundreds of millions of parameters (Sevilla et al., 2022).

These growing complexities and sizes of deep learning models present significant challenges in terms of computation operations (FLOPs), memory footprint, execution time (latency), and energy consumption. As a result, it becomes difficult or even infeasible to store and deploy these models on resource-constrained devices such as edge devices or embedded systems (Neill, 2020). For real-world applications such

as self-driving cars, robotics, and healthcare where real-time performance is crucial, especially the growing model size and the resulting increased inference time become bottlenecks.

Efforts have been made to develop efficient and lightweight models, such as MobileNet (A. G. Howard et al., 2017). Although MobileNet reduces parameters and computational complexity, its execution of depth-wise separable convolutions requires significant memory access, resulting in high energy consumption and reduced efficiency (Sui et al., 2023).

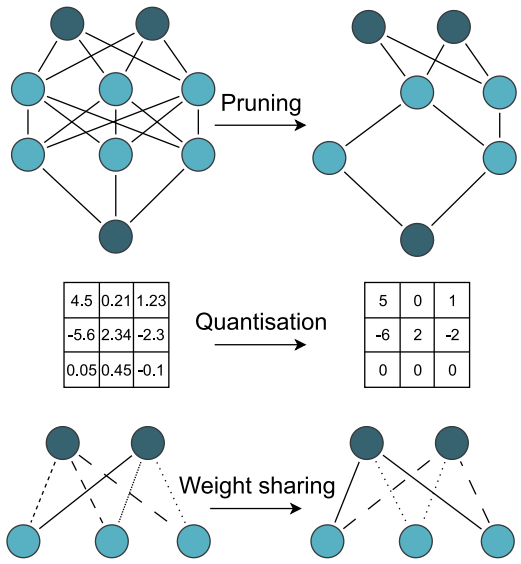

Figure 2.6: Conceptual illustration of some NN Compression techniques. Pruning (top) involves removing redundant and unnecessary connections or structures from the network. Quantisation (middle) is the process of reducing the precision of network parameters by representing them with lower bit representations, while weight sharing (bottom) involves reusing the same weight value across connections or within structures in a neural network (Neill, 2020). In the bottom illustration, different line styles indicate different weights, whereas the same line style indicates the same weights.

To overcome these challenges, alternative approaches have been proposed, including neural network compression techniques like quantisation, weight sharing, and pruning (cf. Figure 2.6). By tailoring the compression strategies to the constraints of each individual hardware, it is possible to substantially reduce the inference latency, memory footprint and energy consumption without affecting the network's performance considerably. It is particularly useful to have structured and robust pruning methods for scaling the computational requirements of a model during development since manually redesigning the model can be too tedious or even infeasible (Cheng et al., 2020; Choudhary et al., 2020; Gale et al., 2019). In this study, our focus is on neural network pruning. It is clear that pruning techniques will continue to play an important role, even as the performance of embedded hardware increases, with an ever-increasing need to deploy bigger and better models while keeping the latency, energy and memory demands within permissible budgets.

2.2 Neural network pruning

Complex tasks often require deep and complex networks with hundreds of millions or even billions of parameters. However, early studies have shown that many of those parameters are redundant and do not contribute much to the model's predictions (Hassibi et al., 1993; Lecun et al., 1989). As a result, it is possible to reduce the size and complexity of the model while maintaining the model's performance, leading to reduced inference time, memory requirements, and perhaps even lower power consumption (Anwar et al., 2015; Gale et al., 2019; Sui et al., 2023). Figure 2.6

illustrates the fundamental concept behind pruning: Pruning is a technique to compress a DNN and involves the systematic identification and removal of redundant or unimportant weights, neurons, or entire structures from a neural network with little or no performance compromise.

Definition Let $W \in \mathbb{R}^d, d \in \mathbb{N}$, describe the parameter matrix of a neural network. Given a mask, $M \in \{0,1\}^d$, pruning of the network can formally be described as the element-wise product of W and M, $W \odot M \in \mathbb{R}^d$. In this context, M is called the pruning mask and $W \odot M$ denotes the parameter matrix of the pruned network.

While the basic idea of pruning is simple - removing unnecessary weights or structures from a neural network - the pruning process can be complex and non-trivial. The selection of weights or structures to remove can significantly impact the final performance of the pruned network. Thus, many different approaches to pruning have been proposed, each with its own strengths and weaknesses.

Approaches to pruning range from simple methods, such as random pruning or magnitude pruning (Han et al., 2015; H. Li et al., 2017), to more complex methods that require a deeper understanding of the underlying principles of neural networks (Hassibi et al., 1993; Lecun et al., 1989; Molchanov et al., 2017). In the following, we introduce classifications of pruning techniques, based on their structure (unstructured vs structured), method (uniform vs non-uniform), and training pipeline (Figure 1.1). Furthermore, we discuss some of the most-relevant pruning techniques and methods.

2.2.1 Pruning structure

Over the past few years, various methods for neural network pruning have been proposed. These approaches can be divided into two main groups: unstructured (Guo et al., 2016; Han et al., 2015; Hassibi et al., 1993; Lecun et al., 1989; Zhu & Gupta, 2017) and structured pruning (Y. He et al., 2018; H. Li et al., 2017; Liu et al., 2017), see Figure 2.7. Unstructured pruning (also called weight pruning) involves removing individual weights from the network, while structured pruning involves removing entire structures, such as neurons from fully connected layers, or filters or channels from convolutional layers.

Unstructured pruning

Lecun et al. (1989) and Hassibi et al. (1993) pioneered unstructured pruning and select redundant and unimportant weights based on their impact on the loss function. These early methods, called Optimal Brain Damage (Lecun et al., 1989) and Optimal Brain Surgeon (Hassibi et al., 1993), estimate weight importance by making a local Taylor approximation of the loss function and using the second-order derivative of each parameter. The main drawback of these methods is that the Hessian matrix needed for the Taylor approximation of the loss is relatively expensive to compute as it has a complexity of $O(n^2)$, where $n \in \mathbb{N}$ is the number of weights in a NN (Liang et al., 2021; Neill, 2020). Because of this, other works have focused on using simpler methods for selecting unimportant weights. For example, Molchanov et al. (2017)

rely only on the first-order Taylor expansions to approximate the impact of different weights on the loss.

One of the simplest weight-pruning strategies is magnitude pruning (Han et al., 2015). In magnitude pruning, it is assumed that weights W_i, $i \in \mathbb{N}^n$ of small magnitude $\|W_i\|_{1,2}$ have a small impact on the performance of a model and thus can be removed. Indeed, under the constraint of L1 or L2 regularisation for example, a penalty term is added to the loss, favouring small weight values. Thus, weights that do not contribute significantly to the model output are expected to shrink during training. Despite their simplicity, magnitude-based pruning strategies are widely used in modern works and have shown excellent performance (Gale et al., 2019; Guo et al., 2016; Renda et al., 2020; Zhu & Gupta, 2017). For example, Han et al. (2015) reduced the number of connections in the CNN architectures AlexNet and VGGNet on ImageNet by 9x to 13x without any loss in accuracy.

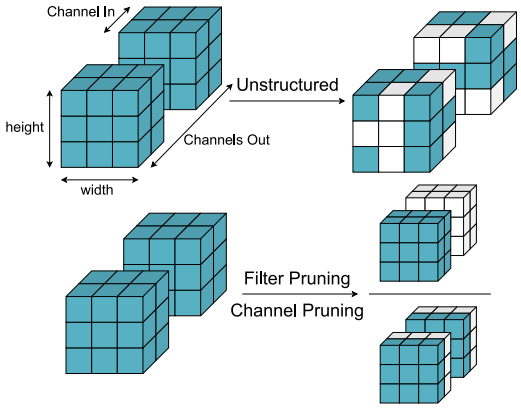

Figure 2.7: Comparison of unstructured (top) and structured (bottom) pruning of convolutional layers. White cubes indicate removed (pruned) weights from the filter/kernel. Adapted from (Sui et al., 2023).

Although unstructured pruning allows for a high level of flexibility and achieves a high sparsity ratio, it may not necessarily result in a more efficient network during inference due to its irregular sparsity. This is because removing individual weights or connections can cause irregular and fragmented networks that are challenging to implement efficiently in hardware, as most hardware is not optimised for sparse matrix multiplications without dedicated libraries (Y. Li et al., 2022; Liu et al., 2019).

Structured pruning

In recent years, structured pruning methods have gained popularity among researchers and practitioners as they effectively speed up models while addressing the challenges of unstructured pruning on resource-constrained devices. By removing entire filters or other structures, the resulting network can be implemented as a dense network, utilising well-defined matrix multiplications of standard libraries. Structured pruning not only reduces the number of parameters but also effectively reduces the dimensions of feature maps, resulting in a model with reduced storage and memory requirements.

In CNNs, typical structured pruning methods involve removing filters or channels of convolutional layers. Figure 2.8 shows the relationship between channel and filter pruning. Pruning a filter in a convolutional layer is equivalent to pruning the corresponding channel in the subsequent layer (Ma et al., 2020): Let $n_i, i \in \mathbb{N}_0$ be the number of input channels for the i-th convolutional layer. Further, let $F_{i,k}, k \in \mathbb{N}_0$

denote the k-th filter of convolutional layer i. Then, each filter of layer i has depth n_i. As shown in Figure 2.8, pruning filter $F_{i,k}, k \in \mathbb{N}_0$ results in the removal of the k-th feature map. Subsequently, removal of the k-th channel for all filters in layer $i+1$, $F_{i+1,j} \forall j \in \mathbb{N}_0$.

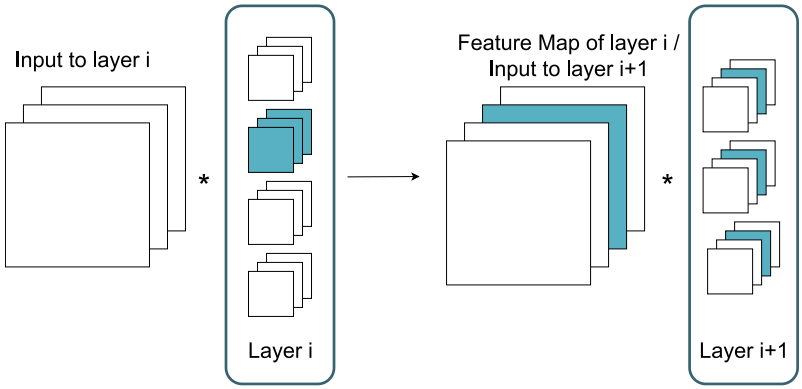

Figure 2.8: Relation between filter and channel pruning in convolutional layers. Blue colour indicates removed/redundant filters or channels. Pruning a filter in layer i is equivalent to pruning the corresponding channels in layer $i+1$. Adapted from (H. Li et al., 2017; Ma et al., 2020).

Furthermore, Figure 2.8 demonstrates an important concept specific to structured pruning: *dependency groups*. Figure 2.8 illustrates that when structures (e.g., channels or filters) in layer i are pruned, it has a cascading effect on other layers. Consequently, the layers influenced by the pruning of structures in layer i are referred to as its dependency group. In simpler terms, pruning a structure in layer i not only impacts that specific layer but also has consequences for other layers in the neural network. The notion of dependency groups helps identify which layers are interconnected and affected by the removal of structures, enabling informed decisions about the pruning process and minimising potential negative effects on the network's performance.

One widely-used approach to structured pruning is the structured counterpart to unstructured magnitude pruning. In structured magnitude pruning, the importance of entire filters is captured according to their kernel weights. For example, H. Li et al. (2017) prune filters with the lowest L1-norm, $\|F_{i,j}\|_1$, and successfully reduce floating-point operations (FLOPs) by 30% on widely-used CNN architectures while maintaining performance. However, recent research has highlighted a concern within the field of pruning. H. Wang et al. (2023) mention "the confusion state network pruning" (p. 1) and criticise the lack of a fair comparison setup in the pruning literature. For example, they re-investigate the effectiveness of L1-norm magnitude pruning and demonstrate comparable or even better performance than more sophisticated pruning techniques by simply modifying the pruning pipeline.

2.2.2 Pruning ratio

One hyperparameter that can have a significant impact on the effectiveness and efficiency of the pruning process, is the pruning ratio (more generally known as the compression ratio). Across the literature, a number of different definitions of

pruning ratio can be found (Blalock et al., 2020). For example in (Liu et al., 2019) the pruning ratio "stands for total percentage of channels that are pruned in the whole network" (p. 6), whereas in (Blalock et al., 2020) "compression ratio is defined as the original size divided by the new size" (p. 8). Additionally, there seems to be no common ground in the choice of pruning ratios for benchmarking, with different works applying different compression ratios (Blalock et al., 2020; Liu et al., 2019; H. Wang et al., 2023).

In this work, we define the pruning ratio in terms of FLOPs. FLOPs describe the number of floating point operations required for a single forward pass. Roughly speaking, ignoring hardware, the higher the FLOPs the slower the model and vice-versa. In the context of pruning, the reduction in FLOPs is a proxy for the reduction of latency, memory usage, and power consumption (Blalock et al., 2020). We define the pruning ratio as the ratio of the total number of FLOPs of the pruned model F_{pruned} to the total number of FLOPs of the original network $F_{original}$,

$$Pruning\ ratio = \frac{F_{pruned}}{F_{original}}.$$

To illustrate, consider a pruning ratio of 0.6. This ratio implies that the model is pruned to 60% of its FLOPs, meaning the resulting compressed model retains 60% of the original FLOPs.

The optimal pruning ratio depends on several factors, such as the underlying pruning method, especially the pruning structure, and the model architecture. For an optimal pruning ratio, the trade-off between a reduction in performance and high compression of the network must be considered, as lower pruning ratios can lead to more significant performance degradation.

2.2.3 Uniform vs non-uniform pruning

During pruning, weights or structures can either be removed on a per-layer (uniform) or a global (non-uniform) level. In uniform pruning, also known as local pruning, the same ratio of, for example, FLOPs or weights is removed in each layer according to the predefined pruning ratio. Non-uniform pruning, sometimes referred to as global pruning, applies the pruning ratio to the whole network and all layers are pruned at once. In non-uniform pruning the layer-level pruning ratio is automatically determined by the pruning algorithm. Figure 2.9 shows a conceptual comparison of uniform and non-uniform pruning.

Uniform layer-wise pruning usually underperforms non-uniform pruning methods (Blalock et al., 2020). This is because different layers may have different levels of importance, and pruning them uniformly can remove important information. Uniform pruning can be considered a baseline each non-uniform pruning strategy should achieve because each pruning mask achieved through uniform pruning can also be achieved through non-uniform pruning.

2.2.4 Pruning pipeline

The standard approach to NN pruning involves the three-step pipeline of training, pruning, and fine-tuning (iterative or one-shot), see Figure 1.1. This three-step pruning pipeline is as follows: first, train the network to the desired level of accuracy; second, prune the network by setting some of the weights to zero based on a predetermined criterion; and third, fine-tune the pruned network to regain the lost accuracy (Han et al., 2015; H. Li et al., 2017).

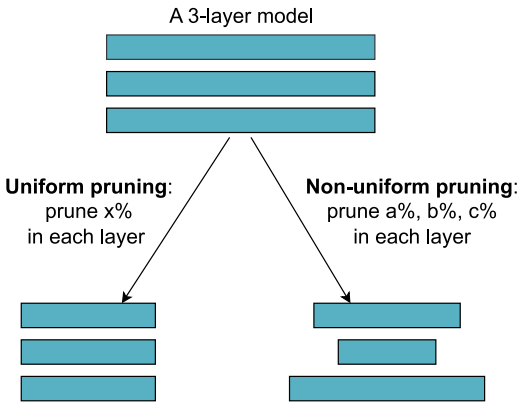

Figure 2.9: Difference between uniform (local) and non-uniform (global) pruning, filter pruning as an example. x is predefined, while a, b, and c are determined by the pruning algorithm. Adapted from (Liu et al., 2019).

However, recent studies have questioned the effectiveness of this pipeline. For structured pruning, Crowley et al. (2019) and Liu et al. (2019) have been able to show that compact models obtained through pruning but retrained from scratch can achieve at least comparable performance compared to fine-tuning the pruned models. In this context, retraining from scratch means training the pruned model as a dense model with randomly initialised weights. Liu et al. (2019) compared a number of structured pruning algorithms, including magnitude pruning, and found that "fine-tuning a pruned model only gives comparable or worse performance than training that model with randomly initialized weights" (p. 1).

Moreover, H. Wang et al. (2023) argue that whether or not retraining from scratch is comparable to fine-tuning a pruned model highly depends on the chosen comparison setup. While "both the pruned architecture and its associated weights are believed to be essential for obtaining the final efficient model" (H. Wang et al. (2023), p. 1), these studies suggest that retraining from scratch may be a viable alternative to fine-tuning in certain scenarios.

In the case of unstructured pruning, Frankle and Carbin (2019) argue

> "dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation reach test accuracy comparable to the original network in a similar number of iterations" (p. 1).

This is called the *lottery ticket hypothesis*. One key aspect when training the subnetwork from scratch is the initialisation of network parameters: Instead of randomly initialising the network, its parameters are reset to their value earlier in the training cycle, see Figure 1.1. The lottery ticket hypothesis has received a lot of attention within the pruning research community as it provides a rationale for pruning without the need for a fully trained model (Lee et al., 2019; Tanaka et al., 2020; C. Wang et al., 2020; Y. Wang et al., 2019). This hypothesis challenges the prevailing belief

that fully-trained over-parameterised networks are necessary for achieving optimal performance (Arora et al., 2018; Chang et al., 2020), and raises the question about how to identify important weights prior to training.

Y. Wang et al. (2019) have investigated the effect of pre-training on pruning in an empirical study. They found that *winning tickets* can be directly derived from a random-initialised over-parameterised dense network with little or no retraining, demonstrating the possibility of pruning at initialisation as displayed in Figure 1.1. The cost of pruning can be reduced by pruning DNNs at initialisation rather than after training. Consequently, this approach presents an attractive alternative to the prevailing and often computationally expensive iterative three-step pruning pipeline. As such, researchers propose pruning at initialisation and aim to further investigate the effectiveness, efficiency, and generalisability of this approach across various network architectures and datasets (Cai et al., 2022; Lee et al., 2019; C. Wang et al., 2020; Y. Wang et al., 2019).

Although pruning approaches differ in how they apply pruning to a model, they share a commonality: pruning is closely tied to the training of a model. A crucial factor in training and optimising DNNs is the choice of model hyperparameters as they affect a model's convergence, generalisability, and overall performance. Properly tuning these hyperparameters is essential to achieve optimal training results and maintain the performance of the pruned model. Therefore, the process of pruning and training a model are linked, with both relying on careful selections of one another for optimal performance and efficiency.

2.3 Hyperparameters in deep learning

Model hyperparameters play a crucial role in deep learning, as they are tightly coupled to the performance of a model during inference and its behaviour during training. Hyperparameters are set before the training process and, unlike model parameters, cannot be automatically learned. Carefully tuning these hyperparameters can significantly improve the performance and robustness of the model (Yu & Zhu, 2020). For example, optimiser-related hyperparameters such as learning rate, momentum, and weight decay influence the speed of convergence and the model's generalisation ability (L. N. Smith, 2018).

However, finding optimal hyperparameters is a challenging and time-consuming task, often requiring years of expertise and long trial-and-error iterations when done manually. Despite the success of deep learning, this process still heavily relies on intuition rather than well-established scientific approaches (L. N. Smith, 2018). The question of how to find the optimal hyperparameters and model architecture in an effective and time-efficient way remains an active area of research with ongoing efforts to improve the overall understanding of neural networks.

In this section, we provide a brief overview of some of the most important hyperparameters, explain their effects on model performance, and explore techniques for optimising them (Bergstra & Bengio, 2012; Jones et al., 1998; L. Li et al., 2018; Mockus, 1975). Additionally, we discuss related work within the hyperparameter

literature, such as studies on the sensitivity of hyperparameters (Novello et al., 2022; Taylor et al., 2021), along with studies on the complex interplay of DL hyperparameters and the impact of architectural changes on them (L. N. Smith, 2018, 2022; S. L. Smith et al., 2018; Yang et al., 2022).

2.3.1 Classification of neural network hyperparameters

We classify deep learning hyperparameters into three types based on their relationship to a model's training, architecture, or optimiser. The following hyperparameter classification is unique to this work, and other works may differ as there is no commonly used hyperparameter categorisation. Table 2.3 provides a brief overview and description of some of the most common training and optimiser hyperparameters.

Hyperparameter	Typical Range or Values	Description
Batch Size	$\{2^n, n \in \mathbb{N}_{\geq 5}\}$	Number of samples in each iteration
Epochs	$[5, 1000]$	Number of training cycles
Learning Rate γ	$[10^{-7}, 0.5]$	Step size of gradient descent optimiser
LR Decay	$[0, 1]$	Reduction of γ
LR Decay Step	$[1, max(epochs)]$	Epoch at which LR decay is applied (during training, LR decay can be applied several times)
Weight Decay λ	$[10^{-7}, 0.5]$	Regularises the model by adding penalty term to the loss
Momentum μ	$[0, 0.99]$	Acceleration factor for gradient descent optimiser

Table 2.3: Description of relevant hyperparameters in this work including typical range or values. Adapted from (Taylor et al., 2021).

Training hyperparameters

Training Hyperparameters, such as the batch size or the number of epochs, control the training process of a model. These hyperparameters are tightly coupled with the speed of training and overfitting of a model. A larger batch size can lead to faster convergence and better utilisation of hardware resources, but it may also result in overfitting or poor generalisation to unseen data (L. N. Smith, 2018). The number of epochs describes the number of times the model iterates over the entire training dataset. A larger number of epochs can lead to better utilisation of hardware but may also result in overfitting or longer training times. Overfitting occurs when the model learns to fit the training data too well, resulting in poor generalisation to new, unseen data (L. N. Smith, 2018).

Architecture hyperparameters

Architecture hyperparameters include parameters related to the model's architecture and weights, for example, the number (depth) or size (width) of layers and their

initialisation. Weight initialisation methods are important in deep learning because they help ensure that the NN can learn effectively during training. Poorly initialised weights can cause the network to take a long time to converge, or not converge at all. Some popular methods include Xavier (also called Glorot initialisation) (Glorot & Bengio, 2010) and He (also called Kaiming) (K. He et al., 2015a) initialisation. These methods take into account the size of the network, the activation functions used, and other factors to help choose the best initial values for the model weights.

Optimiser hyperparameters

In addition to the batch size, number of epochs, and weight initialisation method, optimiser hyperparameters play a crucial role in model training. Stochastic gradient descent (SGD) and its variants, such as Adam (Kingma & Ba, 2017) and Adagrad (Duchi et al., 2011), are commonly used optimizers in deep learning and control how the model's parameters are updated during training. These optimisers have hyperparameters such as the learning rate and momentum that control the update rule for the model's parameters.

The learning rate (LR) is considered one of the most important hyperparameters in deep learning and determines the step size taken in each parameter update (Bengio et al., 2013). As shown in Equation 2.1, this hyperparameter can have a significant impact on training speed and convergence. For instance, a higher learning rate may result in faster convergence, but could also cause the model to overshoot the optimal solution and diverge, while a lower learning rate may converge more slowly but with better accuracy. The LR is tightly coupled with the learning rate scheduler which dynamically adjusting the LR during training. In our experiments, a simple step-wise learning rate schedule is applied. The step-wise LR scheduler decreases the LR at predetermined steps during the training process. Typically, the learning rate decreases towards the end of the training proceeds to fine-tune the model's parameters and to prevent overshooting the optimal solution (Bengio et al., 2013).

Weight decay is a common regularisation technique used in deep learning to prevent overfitting. Weight decay adds a penalty term to the loss function and encourages the model to learn smaller weight values. This penalty term effectively shrinks the weights towards zero during training, reducing their magnitude and making the model less prone to overfitting (Bengio et al., 2013).

When using SGD with momentum as an optimiser, Equation 2.1 shows the weight update after each batch (Paszke et al., 2019). Let γ describe the learning rate, λ the weight decay, and μ momentum which controls the effect of past gradients on the weight update. Let W be the model parameters (weight and biases) we aim to optimise and L the loss of the neural network on the training dataset, then

$$\begin{aligned} V &\leftarrow \mu V + \nabla_W L(W) + \lambda W \\ w &\leftarrow W - \gamma V, \end{aligned} \tag{2.1}$$

where $V \leftarrow \nabla_W L(W) + \lambda W$ at step 0.

Equation 2.1 displays the complex interplay of DNN hyperparameters in training a model. Notably, L. N. Smith (2018) investigate the behaviours of four deep learning

hyperparameters - learning rate γ, weight decay λ, batch size β, and momentum μ - and provide recommendations for setting them. These four hyperparameters are widely recognised as being highly relevant to the training of a model (L. N. Smith, 2018). Previous works such as L. N. Smith (2022) and S. L. Smith et al. (2018) have been able to show an empirical relationship underlining their complex interplay:

$$\frac{\gamma \cdot \lambda}{\beta \cdot (1 - \mu)} \simeq 10^{-6}, \tag{2.2}$$

in the case of SGD as an optimiser.

2.3.2 Hyperparameter optimisation

Hyperparameter optimisation (HPO) refers to the process of finding the optimal hyperparameters of a model for a given dataset and task. The optimal hyperparameters are those that maximise the model's performance, such as its accuracy, or minimise its generalisation error or loss (Yu & Zhu, 2020). Early studies have demonstrated that optimal hyperparameters vary across different datasets and tasks, highlighting the importance of efficient and effective HPO techniques (Kohavi & John, 1995).

Various techniques and toolkits have been introduced to automate the hyperparameter tuning process. These techniques range from simple grid and random search algorithms (Bergstra & Bengio, 2012) to more sophisticated methods like Hyperband (L. Li et al., 2018), which terminates unpromising runs early, and Bayesian optimisation (Jones et al., 1998; Mockus, 1975), which utilises a probabilistic model to approximate the hyperparameter performance space of the network.

Despite these advancements, HPO is still considered a 'black art' due to the complex relationship between hyperparameters and model performance (L. N. Smith, 2018). Moreover, the need for effective and efficient HPO has become more critical than ever given that complex DNNs on large datasets can take days, weeks, or even months to train (OpenAI et al., 2019). Thus, the study of hyperparameters continues to be an important area of research, with ongoing efforts to improve the understanding of hyperparameters and their relationship with a model's performance in order to develop more efficient and effective techniques for optimising complex deep learning models.

Tuning architecture hyperparameters: Neural architecture search

Traditionally, HPO focuses on optimising training and optimiser hyperparameters while keeping the neural architecture fixed. On the other hand, neural architecture search (NAS) automatically searches for the optimal neural architecture, including decisions like the number of layers, nodes per layer, and connections between layers. Historically, HPO and NAS were treated as separate problems, with each addressing their own tasks independently, using specialised techniques and frameworks. However, recent advancements highlight the potential benefits of integrating HPO and NAS into an unified framework. Jointly optimising hyperparameters and architectures enables better performance and more efficient exploration of the search space (Elsken et al., 2019; Shala et al., 2023).

> "The NAS literature at the moment underexplores the impact of hyperparameters on the performance of an architecture. In fact, it is commonly known that the same architecture would perform differently if the training pipeline is altered, for example by changing the learning rate, the number of epochs, or the degree of regularization" (Shala et al. (2023), p. 9)

Although we do not conduct NAS in our study, the process of structured pruning, where the pruned model is retrained after compression, can be considered a form of NAS. In this scenario, pruning identifies the optimal sub-network structure by eliminating structures, such as entire neurons or filters from the network, thereby altering the width of individual network layers and consequently modifying the overall architecture while adhering to the original unpruned network architecture.

2.3.3 Importance of hyperparameters and their optimisation

In classical machine learning (ML), default hyperparameters often provide reasonable performance across a wide range of tasks and datasets. For instance, Weerts et al. (2020) demonstrate that their computed default values for two classical ML algorithms, Random Forest and support vector machines (SVM), often yield comparable performance to tuning hyperparameters. They provide an empirical study using 59 datasets to investigate the performance loss when a hyperparameter is not tuned and set to its default value, investigating whether it is important to tune a hyperparameter. Both works quantify the risk of not tuning a specific hyperparameter, called *tuning risk* in (Weerts et al., 2020) or *tunability* in (Probst et al., 2018).

Although these studies in classical ML have demonstrated that default values often yield satisfactory performance, these findings may not be applicable to deep learning models and tasks. In deep learning, the underlying tasks are typically more complex compared to those where classical ML algorithms excel. Consequently, complex models with numerous parameters are required to tackle the increased complexity of the task and dataset. Given this increased complexity it is non-trivial to generalise optimal hyperparameters across diverse datasets, model architectures, and tasks in deep learning (Yang & Hu, 2022).

However, recent research has shown promising results in transferring optimal hyperparameters between tasks under restricted settings where only a few conditions change. For example, Yang et al. (2022) demonstrate that with *Maximal Update Parameterisation* introduced in (Yang & Hu, 2022) (i) specific deep learning hyperparameters remain stable even as model size changes, and (ii) these hyperparameters can be tuned using a simpler sub-model achieved through scaling the model's width. This technique, referred to as μ*Transfer*, offers the potential for stable hyperparameters across controlled changes to the model architecture.

The approach in Yang et al. (2022) is limited to a few model hyperparameters. For example, hyperparameters related to regularisation, such as weight decay, cannot be easily transferred using their approach. Furthermore, their experiments vary a particular hyperparameter while keeping all other hyperparameters constant, thus overlooking potential interactions between hyperparameters.

Furthermore, Park et al. (2019) studied the effect of network width scaling on the optimal hyperparameters for SGD. Scaling the width of a model can be seen as uniformly pruning (or extending) a model. For example, in CNNs, width scaling a model refers to uniformly, with respect to the original model size, adding or removing layers from convolutional layers. Park et al. (2019) showed a proportional relation between the learning rate and batch size and network width, in the absence of batch normalisation.

We contribute to the existing research by addressing the behaviour of hyperparameters under changes caused by pruning. Specifically, we focus on interactions between learning rate and weight decay, as they are known for their strong relationship, see Equation 2.2.

Importance of hyperparameters

Usually, not all hyperparameters are of equal importance to the performance of a model. Meaning, that by tuning just a few hyperparameters, model performance can be significantly improved. As a result, there are a few studies on the ranking of hyperparameters.

For example, Novello et al. (2022) combine hyperparameter optimisation with a sensitivity analysis of hyperparameters. In general, sensitivity analysis involves capturing the effect of inputs on a function's output. Their approach employs a statistical dependence measure to quantify hyperparameter contributions to model performance, resulting in a more interpretable HPO process that concentrates on the most relevant hyperparameters. Furthermore, Taylor et al. (2021) employ sensitivity analysis to assess the relationship between model architecture, hyperparameter convergence, and input data statistics. They show that the influence of a hyperparameter is closely tied to the architecture, with shallower models more strongly influenced by training hyperparameters, such as the number of epochs, and deeper models being more strongly influenced by optimiser hyperparameters. In this context, Taylor et al. (2021) define the influence of a hyperparameters "in terms of contribution to model accuracy" (p.1).

2.3.4 Hyperparameter performance space

In this section, we formally introduce the concept of the hyperparameter performance space. In our experimental setup, we focus on two specific model hyperparameters: learning rate and weight decay. Both of these hyperparameters are continuous in nature. For the sake of simplicity and clarity, we concentrate on the two-dimensional scenario, where only two continuous hyperparameters can be adjusted. The following definition of the hyperparameter performance space can be easily adapted to any number of continuous hyperparameters.

Definition For $k \in \mathbb{N}$, let $\mathcal{X}_k$ describe the space of possible values of the k-th hyperparameter of a neural network. Further, let the i-th and j-th hyperparameters of the model be flexible, with $i, j \in \mathbb{N}$, and all other hyperparameters are kept fixed. We define $f : \mathcal{X}_i \times \mathcal{X}_j \to \mathbb{R}$, where $f(x, x')$ describes the

performance of the model on a held-out validation set after training with the hyperparameters $x \in \mathcal{X}_i,\ x' \in \mathcal{X}_j$. Then, $\{(x, x', f(x, x')) \mid (x, x') \in \mathcal{X}_i \times \mathcal{X}_j\}$ describes the hyperparameter performance space of the model with respect to the i-th and j-th hyperparameter.

In our experiments, we approximate the relationship between hyperparameters and performance under pruning by sampling from the joint space of $\mathcal{X}_i \times \mathcal{X}_j$ and training the model with the given hyperparameter initialisation. To gain a deeper understanding of this space, we use a number of tools and methods. These include visual analysis and interpretation techniques, as well as quantified measures. By employing these approaches, we can effectively compare the performance landscape across various pruning strengths and methods, gain insight into the impact of pruning on this relationship, and assess the risk of HPO after pruning.

Chapter 3

Methodology

As discussed in Section 2.3, hyperparameters and their interaction with changes in the architecture, dataset, or task have been widely studied in the deep learning literature. Numerous studies have investigated the importance of selecting appropriate hyperparameters to achieve optimal model performance. The architecture of a model is one factor that significantly affects the selection of hyperparameters (cf. Section 2.3). Different architectures may require different hyperparameter settings to attain the best results. To our best knowledge, the impact of pruning, which alters the architecture of a model in a structured way, on the optimal hyperparameters has received limited attention in the literature.

In this chapter, we discuss our methodology and experimental setup for investigating the impact of deep neural network pruning on the hyperparameter performance space of a model. The experimental design is structured as follows: Given a dataset-model combination with optimal hyperparameters (Section 3.1), we first span a bigger region of the hyperparameter performance space around the vicinity of the optimal hyperparameter set (Section 3.3) in order to approximate the performance surface we introduced in Section 2.3. Next, we prune the models to various degrees (Section 3.2) utilising widely-used structured pruning methods: uniform and non-uniform magnitude pruning. Then, the hyperparameter performance space of the compressed model is captured by retraining this model from scratch, following pipeline (d) in Figure 1.1. By observing the shift in the performance landscape between the original model and pruned model, we aim to identify how hyperparameters change under pruning and how aggressively models can be pruned before the hyperparameters need to be reconsidered for optimal performance.

The experiments in our work are designed in a hierarchical order, from simple to more complex. Our experiments vary in terms of dataset size (CIFAR-10 and ImageNet), model architecture and complexity (ResNet and MobileNet), and pruning method (uniform and non-uniform magnitude pruning), allowing for a comprehensive study of the effect of structured magnitude pruning under different conditions.

3.1 Datasets, model architectures, and training setup

In our experiments, we follow recommendations given by Blalock et al. (2020) and utilise two popular image recognition datasets in our experiments: CIFAR-10 (Krizhevsky, Hinton, et al., 2009) and the image classification dataset from the large-scale visual recognition challenge 2012-2017 (ImageNet) (Russakovsky et al., 2015); as well as three widely used CNN architectures for image classification: ResNet-56, ResNet-50 (K. He et al., 2015b), and MobileNetV2 (Sandler et al., 2019), see Section 2.1.1.

3.1.1 Datasets

Both datasets, CIFAR-10 and ImageNet, are popular benchmarks in computer vision and have been widely adopted in the pruning and deep learning communities (Blalock et al., 2020). Figure 3.1 shows a random selection of training images in the CIFAR-10 and ImageNet datasets. CIFAR-10 is a smaller and simpler dataset compared to ImageNet, with low-resolution images and a limited number of classes, see Table 3.1. Therefore, it provides a faster and less computationally expensive way to evaluate the impact of different pruning techniques and hyperparameters on the performance of a model. On the other hand, ImageNet is a much larger and more complex dataset with higher-resolution images and a larger number of classes. In comparison to CIFAR-10, ImageNet provides a more challenging and realistic classification task, closer to real-world scenarios (Russakovsky et al., 2015).

Dataset	Input shape	Training Images	Validation Images	Classes
CIFAR-10	$32 \times 32 \times 3$	$50,000$	$10,000$	10
ImageNet	$224 \times 224 \times 3$	$1,281,167$	$50,000$	1000

Table 3.1: Comparison of CIFAR-10 and ImageNet datasets. The table displays the image dimensions as used in this work.

CIFAR-10 CIFAR-10 consists of $60,000$ 32×32 colour images in 10 classes. The dataset is split into $50,000$ training and $10,000$ validation images, with a balanced distribution of classes in both sets. CIFAR-10 has been widely used to evaluate the performance of various DL models and pruning methods due to its limited size and simplicity (Blalock et al., 2020; Krizhevsky, Hinton, et al., 2009).

ImageNet ImageNet provides a diverse set of images representing various objects, animals, and scenes in 1000 different categories, thus introducing more diversity compared to CIFAR-10. The dataset comprises over 1.2 million training images and $50,000$ validation images, with a balanced distribution of images across the classes. The balanced distribution of images across the 1000 classes ensures that no single class dominates the dataset, making it a fair evaluation of the performance (Russakovsky et al., 2015).

(a) CIFAR-10 (b) ImageNet

Figure 3.1: Randomly selected training images of the CIFAR-10 and ImageNet datasets. For CIFAR-10 the images are resized to dimensions 32×32.

Utilising both the CIFAR-10 and ImageNet datasets allows us to conduct a comprehensive analysis. This approach takes into account that the optimal hyperparameters depend not only on the architecture of a model but also on the specific task and dataset, as discussed in Section 2.3. Evaluating the impact of pruning across varying complexity, size, and image resolution increases the robustness of our results.

For example, if we obtain consistent results across both datasets, we can assume that the observed effects are more likely to hold in other similar datasets or real-world applications. Therefore, utilising two different datasets increases the validity and reliability of our findings. Conversely, inconsistencies between the datasets can also provide insights into the limitations and specific characteristics of the impact of pruning on a model's hyperparameter landscape.

Our experimental setup involves two distinct data subsets: the training and validation datasets. Given a hyperparameter configuration, the training dataset is used by the optimiser to train the model. Once the training is completed, we evaluate the model's performance on the validation dataset. The performance of the model on the validation set, along with the corresponding hyperparameter configuration, constitutes a single point in the hyperparameter performance space.

Data preprocessing

Data preprocessing, particularly data augmentation, is crucial for deep learning as it increases the number of training examples and introduces variety. Data augmentation involves altering the training data, which makes it more challenging for the model to memorise the data and encourages the learning of general patterns. By training on augmented data, the model becomes more robust and capable of handling variations in real-world data, often leading to improved performance and generalisation.

Our data augmentation pipeline for CIFAR-10 follows widely-adopted practices (K. He et al., 2015b): To prepare the training set, a zero-padding of 4 pixels is added to each side of the image. Next, a 32×32 crop is randomly sampled from the padded image or its horizontal flip, and finally, the image is normalised by subtracting the mean and dividing it by the standard deviation of the training dataset. This normalisation step ensures that the pixel values have a zero mean and a standard deviation of one, which is a common practice in image processing for deep learning. For the validation set, we only apply normalisation to the single view of the original 32×32 image. For both, the training and validation subsets, the normalisation is based on the training datasets' mean and standard deviation.

For training on ImageNet, we preprocess the images by randomly sampling a 224×224 crop from the resized 256×256 colour image, followed by normalisation of the single view or its horizontal flip, adopting common practices (K. He et al., 2015b). For validation, images are only centre-cropped to size 224×224 and normalised. Similar to the normalisation process used for the CIFAR-10 dataset, normalisation of the ImageNet training and validation set involves subtracting the mean (of the training dataset) and dividing by the standard deviation of the training dataset.

With this adaptation of the original MobileNetV2 architecture we follow the from Ayi and El-Sharkawy (2020) as used for their basemodel.

3.1.2 Model architectures

All three architectures we utilise in our experiments, namely ResNet-56, ResNet-50 (K. He et al., 2015b), and MobileNetV2 (Ayi & El-Sharkawy, 2020; Sandler et al., 2019), are commonly used in the DNN pruning literature (Blalock et al., 2020; Liu et al., 2019). Specifically, we utilise an adapted version of MobileNetV2 for $32 \times 32 \times 3$ images as described in Section 2.1.1 and ResNet-56 on the CIFAR-10 dataset, as well as ResNet-50 on the ImageNet dataset. These architectures were chosen based on their proven track record of success in previous literature and their ability to handle the complexity of the datasets used in this study. Additionally, we intentionally included a wide range of model complexities to encompass a broad spectrum of architectural designs and capabilities. Table 3.2 summarises the key characteristics of the architectures.

Architecture	Input shape	Parameters (M)	FLOPs
ResNet-56	$32 \times 32 \times 3$	0.85	2.5×10^8
MobileNetV2	$32 \times 32 \times 3$	2.2	6.1×10^8
ResNet-50	$224 \times 224 \times 3$	25.6	8.2×10^9

Table 3.2: Comparison of MobileNetV2, ResNet-56, and ResNet-50. The number of FLOPs are measured using "Embedl Model Optimization SDK" (2023).

The purpose of using multiple model architectures in our experiments is to get a better understanding of the factors influencing model performance and their interactions under pruning. ResNets are widely recognised for their ability to effectively train

very deep neural networks using residual connections. In contrast, MobileNetV2 is characterised by its computational efficiency, making it suitable for deployment on resource-constrained devices. By using both ResNets and MobileNetV2, we gain valuable insights into the transferability and generalisability of our results across varying model depths and architectural designs.

For example, if we observe that pruning has a consistent impact on the performance landscape across various model architectures, it suggests that the effect of pruning may be less dependent on the architecture. Instead, its effect could be more influenced by the characteristics of the task and the pruning method employed.

3.1.3 Training setup and model hyperparameters

In our experiments, we utilise the Python deep learning library PyTorch for training and evaluation of models (Paszke et al., 2019). Model implementations and training recipes are adapted from various sources:

- Idelbayev (2018): Implementation of ResNet-56 and training recipe for the experiments on CIFAR-10
- Chen (2018): Implementation of MobileNetV2
- maintainers and contributors (2016): Implementation of ResNet-50 and training recipe for the experiments on ImageNet

On CIFAR-10, we use the same training and initialisation setup for ResNet-56 and MobileNetV2, shown in Table 3.3. In particular, we follow descriptions from K. He et al. (2015b) closely and train both models with SGD as an optimiser with momentum set to 0.9. We initialise the model parameters via He initialisation before training and before retraining from scratch after pruning. ResNet-56 and MobileNetV2 are trained with a batch size of 128 using a single NVIDIA T4 or A40 GPU, respectively. Both models are trained for 180 epochs, with a step-wise learning rate schedule dividing the learning rate by 10 after 50% and 75% of the training epochs (after 90 and 135 epochs). This training setup follows widely-adopted practices for training MobileNetV2 and ResNet-56 on the CIFAR-10 dataset.

Hyperparameter	Value for CIFAR-10	Value for ImageNet
Weight Initialiser	He Initialisation	He Initialisation
Optimiser	SGD	SGD
LR Schedule	step-wise	step-wise
LR Decay	0.1	0.1
LR Decay Steps	90, 135	30, 60
Momentum	0.9	0.9
Batch Size	128	256
Epochs	180	90

Table 3.3: Training setup and model hyperparameters for ResNet-56 and MobileNetV2 on CIFAR-10 and for ResNet-50 on ImageNet. See Table 2.3 for descriptions of the hyperparameters.

For training ResNet-50 on ImageNet we adopt the simple training recipe outlined in the Python library Torchvision (maintainers & contributors, 2016), see Table 3.3. Similar to the training of ResNet-56 and MobileNetV2 on CIFAR-10, ResNet-50's optimiser for training is SGD with momentum set to 0.9 and its weights are initialised using He initialisation. ResNet-50 is trained with a mini-batch size of 256 utilising a single NVIDIA V100 GPU. Training is terminated after 90 epochs and the learning rate is divided by 10 every 30 epochs.

Performance metrics

To understand the relationship between learning rate, weight decay, and the pruning ratio, we train a model for varying values of learning rate, weight decay, and pruning ratio and measure final performance on held-out validation data.

Top-1 accuracy For evaluating a model's performance on the validation set we use the top-1 accuracy as a metric: Let $\hat{y}$ be the predicted values of the samples in the validation dataset with true values y, then the top-1 accuracy is defined as

$$\text{top-1 acc}(y, \hat{y}) = \frac{100}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \mathbb{1}_{\{y_i\}}(\hat{y}_i), \tag{3.1}$$

where $n_{samples} \in \mathbb{N}$ is the number of samples in the validation set and $\mathbb{1}$ is the indicator function; for example, given a set A it is $\mathbb{1}_A(x) = \begin{cases} 0 & \text{if } x \in A, \\ 1 & \text{if } x \notin A. \end{cases}$. All of our datasets have a balanced distribution of classes in the validation subset. Hence, the top-1 accuracy poses a fair evaluation of the performance of the model.

Top-5 accuracy In the case of experiments on ImageNet we additionally report the top-5 accuracy:

$$\text{top-5 acc}(y, \hat{y}) = \frac{100}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \sum_{j=1}^{5} \mathbb{1}_{\{y_i\}}(\hat{y}_{i,j}), \tag{3.2}$$

where $\hat{y}_{i,1}, \ldots, \hat{y}_{i,5}$ indicate the top-5 predicted classes of the i-th sample in the validation set. The top-5 accuracy is a common method for benchmarking models on ImageNet because this dataset contains a large number of classes (1000 different classes) with many being visually similar.

While the top-1 accuracy measures whether the model's top predicted class matches the ground truth label, considering only the most confident prediction. The top-5 accuracy considers whether the correct label is in the model's top-5 predictions. Due to the complexity of ImageNet, there can be cases where the model's prediction is still considered reasonable even if the top prediction is incorrect. For example, if the ground truth label is'greater swiss mountain dog,' but the model predicts 'bernese mountain dog' as the top class, it's still a reasonably accurate prediction (Russakovsky et al., 2015), see Figure 3.1b. Thus, by reporting both top-1 and top-5 accuracy for ImageNet experiments, we get a more comprehensive understanding of the model's performance on this dataset.

We do not report the top-5 accuracy for models on the CIFAR-10 dataset. This dataset is relatively simple with 10 distinct classes (Krizhevsky, Hinton, et al., 2009). Because of this, reporting only the top-1 accuracy is considered sufficient in research and practice as the additional information provided by the top-5 accuracy is less meaningful given the small number of classes in this dataset.

3.2 Pruning method

As discussed in Section 2.2, there exists a wide range of pruning techniques and methods to choose from. The goal of this study is to gain a better understanding of the relationship between pruning and model performance in general, rather than focusing on achieving state-of-the-art results. Hence, we select well-established techniques for pruning and utilise uniform (local) and non-uniform (global) structured magnitude pruning, according to the L1-norm of the weights of the filters, $\|F_{i,j}\|$, $i, j \in \mathbb{N}_0$, given the notation introduced in Section 2.2. For pruning, we employ the "Embedl Model Optimization SDK" (2023).

The non-public availability of the Embedl Model Optimisation SDK may raise concerns regarding the reproducibility of results. Since the SDK is not accessible to the wider research community, it restricts the ability to replicate and verify the findings of our study. However, our experiments rely on widely-adopted and well-known pruning techniques that can be easily implemented using alternative open-source libraries or frameworks. In the following, we provide a description of the pruning methodology employed in our study, referring to relevant literature to ensure transparency and facilitate reproducibility to the best extent possible.

Magnitude pruning is a simple yet widely-used and effective technique for compressing a model. It has proven to be a staple in the pruning literature and is often considered a baseline for many modern and more complex pruning methods, as we discussed in Section 2.2.1. Its simplicity makes it easy to implement and interpret, while its effectiveness in both structured and unstructured pruning scenarios has been well-documented, see Section 2.2.1. By selecting this widely-used technique, we aim to gain valuable insights into the general relationship between pruning and model performance.

In the remainder of this sections, we provide a detailed explanation and rationale of how we prune. Specifically, we follow the classification introduced in Section 2.2 and discuss the structure (unstructured vs structured), method (uniform vs non-uniform), and training pipeline of our pruning method. Further, we present the specific pruning ratios employed in our experiments. The choice of pruning ratio plays a significant role in determining the strength of pruning, which, in turn, has a crucial impact on the final performance and architecture of the model (cf. Section 2.2.2).

3.2.1 Pruning structure

In our experiments, we focus on structured pruning methods due to their widespread interest in both research and practice. In contrast to unstructured pruning, structured pruning results in a dense network, allowing for better hardware utilisation and improved inference speed (cf. Section 2.2.1).

As shown in Figure 2.8, filter and channel pruning can be used interchangeably: Pruning an input channel of a layer is the same as pruning the corresponding filter of the previous layer and vice-versa. Thus, in the context of this work, we don't differentiate between filter and channel pruning for convolutional layers. Instead, the removal of filters or channels is collectively referred to as structured pruning.

3.2.2 Pruning ratio

As discussed in Section 2.2.2, the pruning ratio is one of the most important hyperparameters when pruning a model. We expect different pruning strengths (as determined by the pruning ratio) to have different effects on the hyperparameter performance space of a model. This is because a higher pruning ratio produces a pruned model closer to the original model compared to a smaller pruning ratio.

For experiments on CIFAR-10 we perform uniform and non-uniform magnitude pruning using two different pruning ratios for each model. Namely, for ResNet-56 on CIFAR-10 we utilise pruning ratios of 0.6 and 0.2, meaning that 60% or 20% of the model's FLOPs remain after pruning. For MobileNetV2 on CIFAR-10 we apply pruning with a strength of 0.6 and 0.4. These choices allow us to capture the effects of moderate and more aggressive pruning on the models.

We chose these pruning ratios for the following reasons: First, a pruning ratio higher than 60% would likely have a limited impact on the model. Pruning ratios of 60% and above might result in minimal changes to the model's structure or performance. By selecting 60% as the higher pruning ratio, we can assess the effects of moderate pruning while still expecting noticeable differences compared to the unpruned model. Second, the lower pruning ratio of 20% for ResNet-56 and 40% for MobileNetV2 leaves a reasonable number of channels intact. Thus, the pruning ratios 60% and 20% (ResNet-56) and 60% and 40% (MobileNetV2) provide a reasonable range of pruning strengths to evaluate.

For ResNet-50 we choose only a single pruning ratio of 0.4 because training on the ImageNet dataset takes substantially more time compared to training on CIFAR-10. While training ResNet-56 on CIFAR-10 takes at most two hours with our setup allowing for relatively fast comparison, the training of ResNet-50 on ImageNet takes roughly 2 days due to the increased complexity of the ImageNet dataset (higher resolution and size) and model (increased number of parameters and FLOPs) compared to the other models and dataset (cf. Table 3.1 and Table 3.2). Further, a pruning ratio of 40% ensures reasonable performance while still removing a significant part of the network.

3.2.3 Uniform vs non-uniform pruning

In our experiments, we explore the impact of the pruning method on the hyperparameter performance space by applying both uniform and non-uniform structured pruning. Specifically, for experiments conducted on CIFAR-10, we employ uniform and non-uniform structured magnitude pruning with the same pruning ratio. This allows us to compare the effects of both methods on the hyperparameter performance space.

However, for experiments conducted on ImageNet, we solely focus on non-uniform structured magnitude pruning. This decision is driven by two factors: Firstly, training networks on ImageNet is extremely demanding in terms of time and resources. To optimise resources, we prioritise non-uniform pruning for the ImageNet experiments. Secondly, in practical scenarios, non-uniform pruning methods are more commonly employed as they offer greater flexibility. Consequently, studying the impact of non-uniform pruning becomes more significant and relevant to real-world applications.

While non-uniform pruning is more interesting from a practical point of view as it allows for more flexibility, the comparison of the effect of non-uniform and uniform pruning can lead to meaningful conclusions. For example, if we find the effect of pruning to be independent of the pruning method (uniform or non-uniform), many works investigating the effect of model width scaling on the hyperparameter space can potentially be transferred to the more general non-uniform pruning case.

3.2.4 Pruning pipeline

In our experimental setup, we deviate from the conventional three-step approach of training, pruning, and fine-tuning. Instead, we opt to retrain the pruned model from scratch, as shown in Figure 1.1. By choosing to retrain the pruned model from scratch, we introduce a distinct approach to evaluating the effect of structured pruning on a model.

We have chosen this pipeline for several reasons: As discussed in Section 2.2.4, there is no standard in the literature, and other pruning pipelines may introduce additional hyperparameters. For instance, in the traditional three-step approach the length of the fine-tuning step or, in the rewinding approach the epoch t at which weights or other hyperparameters are rewound to are introduced as additional hyperparameters (cf. Figure 1.1). These hyperparameters can make the experimental setup more complex and difficult to compare. Recent research has brought these issues to attention, highlighting the lack of a generalised experimental setup (H. Wang et al., 2023). The simple and straightforward approach of retraining from scratch does not introduce any additional hyperparameters. This reduces the complexity and potential sources of error and makes the experimental setup more streamlined and easier to reproduce. Furthermore, retraining the pruned model from scratch allows for a fresh optimisation process. It ensures that the pruned model is trained with a refined weight initialisation, considering the reduced model complexity resulting from pruning.

3.2.5 Pruning ResNets and MobileNets

Pruning Residual and Mobile networks is not straightforward due to their skip connections, requiring the input of a connected block to have the same number of channels (depth) as its output, as illustrated in Figure 2.3 and Figure 2.5. For pruning skip connections, pruning is conducted only within blocks, following L1-norm magnitude pruning (H. Li et al., 2017). Furthermore, all fully-connected layers and the first convolutional layer in ResNets are spared, as is common practice (Gale et al., 2019; H. Wang et al., 2023; H. Wang et al., 2021; Zhu & Gupta, 2017).

3.3 Hyperparameter performance space

In our study, we focus on the impact of pruning on two hyperparameters, learning rate and weight decay. The learning rate is one of the most important hyperparameters when training DNNs as discussed in Section 2.3. Further, there is a complex relationship between the strength of regularisation as imposed through weight decay and the learning rate of a model. We aim to capture this relationship and how it might be affected by pruning by investigating the impact of pruning on the learning rate and weight decay in conjunction.

In our experiments, we approximate the relation between learning rate, weight decay, and performance of a model by spanning a grid around the vicinity of the *prior*. In this context, the prior describes the pre-defined or well-known hyperparameter settings of learning rate and weight decay for the given model and task as reported in research or well-known libraries. Table 3.4 shows the prior values of learning rate and weight decay for ResNet-56 and MobileNetV2 on CIFAR-10 and ResNet-50 on ImageNet as used in our experiments. Prior values for ResNet-56 and ResNet-50 have been reported in the original work (K. He et al., 2015b).

Architecture	Prior Learning Rate	Prior Weight Decay	Baseline Top-1 Acc	Baseline Top-5 Acc
ResNet-56	$10^{-1.0}$	$10^{-4.0}$	93.340	-
MobileNetV2	$10^{-1.0}$	$10^{-4.0}$	94.730	-
ResNet-50	$10^{-1.0}$	$10^{-4.0}$	75.448	92.608

Table 3.4: Prior learning rate and weight decay and baseline accuracies for ResNet-56 and MobileNetV2 on CIFAR-10 and ResNet-50 on ImageNet.

For training the modified version of MobileNetV2 on CIFAR-10 a number of different hyperparameter configurations and training recipes have been reported. For example, kuangliu (2017) train MobileNetV2 on CIFAR-10 with a learning rate of $10^{-1.0}$ and a weight decay of $5 \times 10^{-4.0}$, following a similar training recipe to our work. They achieve a top-1 accuracy of 94.43%, slightly lower than our baseline accuracy of 94.73% as shown in Table 3.4. Further, Chen (2018) and Ayi and El-Sharkawy (2020) report top-1 accuracies of 94.71% and 94.3% on CIFAR-10, respectively. In both works different training recipes and hyperparameter settings are used. We chose the prior learning rate of $10^{-1.0}$ and weight decay of $10^{-4.0}$ for MobileNetV2 for two

reasons: First, we are not interested in a single point but rather the overall shape and second, all reported weight decay learning rate configurations for MobileNetV2 are within the vicinity of our chosen prior and, thus, will be captured in our experiments.

Note, we do not perform hyperparameter optimisation to gain optimal initial (prior) values of learning rate and weight decay. This is because, HPO can be computationally expensive and time-consuming, as most common approaches to HPO require training and evaluating the given model multiple times. Further, for our selected models there is already extensive research and practical experience available regarding suitable hyperparameter settings for the CIFAR-10 and ImageNet datasets (K. He et al., 2015b; maintainers & contributors, 2016; Sandler et al., 2019). By using previously reported hyperparameter settings, we allow our study to be comparable to other works in the area of pruning and NN compression. In many works on pruning, researchers aim to compare their results with existing literature to demonstrate the effectiveness of their proposed methods. By using the same or similar hyperparameter settings as previous studies, our experiments can be directly compared to those works, ensuring a fair and meaningful evaluation of our approach.

In the deep learning literature, it is common practice to evaluate and train models multiple times on the same hyperparameters to account for the inherent randomness and variability in the training process. For example, the initial random initialisation of weights can lead to different starting points in the optimisation landscape, resulting in different convergence paths and final performance. However, in our experiments, we only train and evaluate a model on the given weight decay and learning rate grid once. This choice is on the one hand side attributed to the time constraints of this work. On the other hand side, we are not interested in the average performance of the model on a single point of the grid but rather in the overall shape of the performance landscape. Small variations will have a neglecting effect on the overall shape of this space.

For the grid, we log-scale learning rate and weight decay. This is to provide a smoother and more balanced exploration of the hyperparameter space as the increments between values are more evenly distributed on a log-scale for learning rate and weight decay. For experiments on CIFAR-10 we span an initial grid of 9×9 around the prior with a step size of 0.2 (log-scale) and on ImageNet we span a smaller grid of 3×3 with a step size of 0.4 (log-scale). Depending on the results on the initial grid, we extend the grid to capture most of the optimal hyperparameter area which we define below.

As mentioned, we are interested in the area of optimal hyperparameters. We define this area with respect to a model's top-1 accuracy as a performance measure. Let Θ describe the set of weight decay and learning rate combinations for the given model in our experiments. Then it is $\Theta \subseteq \Lambda \times \Gamma$, where Λ describes the space of possible weight decay values and Γ describes the space of possible learning rate values. Following the notations introduced in Section 2.3.2, we define $f : \Lambda \times \Gamma \to [0, 100]$, where $f(\lambda, \gamma)$ describes the top-1 accuracy of the model on a held-out validation set after training. The region of interest (ROI) given a threshold $x \in \mathbb{R}$ is defined as

$$\text{x-ROI} := \{(\lambda, \gamma) \mid (\lambda, \gamma) \in \Theta,\ f(\lambda, \gamma) \geq f(\theta^*) - x\} \subseteq \Theta, \tag{3.3}$$

where $\theta^* \in \Theta$ describes the hyperparameter configuration that maximises the performance of the model, $\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}} f(\theta)$. For example, 0.5-ROI is the set of the weight decay and learning rate combinations for which the model's performance is within the top 0.5% given its best-achieved top-1 accuracy $f(\theta^*)$. We note, for x $\leq$ y it is x-ROI $\subseteq$ y-ROI.

3.3.1 Approach to Q1: Effects of pruning

To gain valuable insights from the collected data in regard to the first research question[1], we analyse the data using various methods. We only discuss top-1 accuracy as a performance measure in this section; all concepts can be easily transferred to top-5 accuracy.

Visual analysis

First, we perform a visual analysis of the hyperparameter performance space under the different pruning conditions. Through visualisations we are able to determine the optimal hyperparameter areas and get an intuition for how they might change under pruning.

For visually interpreting the data, we utilise the graphing-library Plotly (Inc., 2015). Specifically, we are dealing with 3-dimensional data (weight decay, learning rate, top-1 accuracy) and use contour and scatter plots for visualisation.

In both the scatter and the contour plots, the 3-dimensional data is represented in a 2-dimensional format on the (weight decay, learning rate)-plane. For scatter plots, the colour of data points describes the top-1 accuracy, whereas in contour plots the top-1 accuracy is depicted using contour lines and colour gradients, allowing for a smoother visualisation. The contour lines in the contour plot represent the surface of the hyperparameter performance space by connecting hyperparameter settings for which the model achieves similar performance. Further, the distance between individual contour lines describes the gradient of the underlying surface: when the lines are close together, the gradient's magnitude is high, whereas when the lines are farther apart, the gradient's magnitude is lower. This is because the difference in performance between two lines is always the same (Inc., 2015).

The scatter plot provides a realistic representation of the data, displaying individual data points. On the other hand, the contour plot presents a smooth visualisation of the surface by interpolating between data points. Despite the interpolation, the contour plot still provides a close approximation of the optimal hyperparameter area to the actual experimental data. This is mainly due to two reasons: (i) our chosen step size is relatively small, ensuring a fine-grained representation, and (ii) we will find that the landscape appears to be relatively smooth with minimal irregularities and outliers, as observed in the scatter plot.

[1] What are the effects of structured pruning on the hyperparameter performance space of a model?

Quantitative analysis

In addition to visually interpreting the data, we quantify the effects of pruning on the optimal hyperparameter area of a model. Specifically, we consider the 0.5-ROI and 1.0-ROI with respect to the model's top-1 accuracy. Given a hyperparameter configuration, the performance of the model with this configuration is subject to small variations due to randomness in the training and initialisation process. Thus, the x-ROI describes a random sample of the underlying distribution of optimal hyperparameters. Thus, instead of investigating the absolute changes of x-ROI under pruning, we are interested in how the overall shape of this region changes.

Given the visual analysis, we will find that for both samples, 0.5-ROI and 1.0-ROI, a Gaussian model is appropriate. More specifically, we utilise confidence regions to model the data.

Intuitively, the confidence region can be described as follows: Given a confidence level α and a population parameter of interest (for example, the population mean) then in $100(1-\alpha)\%$ of samples the $100(1-\alpha)\%$ confidence region captures the true value of the population parameter (Neyman, 1937). In the case of a bivariate Gaussian distribution, the confidence region describes the contour line of the probability density function of the underlying Gaussian distribution and is centred at the sample's mean. In this case, the confidence region is called the confidence ellipse as its shape is elliptic. For example, given a confidence level of 0.05, the 95% confidence ellipse describes the region in which 95% of samples drawn from the underlying Gaussian distribution fall. Consequently, the confidence ellipse represents the standard deviations and is also called the standard deviation ellipse (SDE) (B. Wang et al., 2015).

Table 3.5 shows the confidence level given the SDE multiplier. For example, the 86.47% confidence ellipse corresponds to the contour line of the probability density function of the underlying Gaussian distribution at 2 multiples of the SDEs.

SDE multiplier	confidence level
1	0.6065
2	0.1353
3	0.0111
4	0.0003

Table 3.5: Confidence levels of the scaled standard deviation ellipse (SDE). Adapted from (B. Wang et al., 2015).

We quantify the confidence ellipse of 0.5-ROI and 1.0-ROI at confidence level $\alpha = 0.1353$ using measurements such as its length, width, and mean. The resulting ellipses describe the region in which 86.47% of optimal hyperparameter configurations fall given the assumptions above.

3.3.2 Approach to Q2: Performance risk

From the analyses of the first research question, we gain a good intuition for the second research question[2]. We quantify the performance risk following a similar approach to Probst et al. (2018) and Weerts et al. (2020).

[2]Q2: What is the performance risk when not tuning hyperparameters after pruning?

First, we introduce the following notations given a model and a dataset (adapted from (Probst et al., 2018; Weerts et al., 2020)). Adapting the notation introduced in Section 2.3, let $f(\lambda, \gamma) \in \mathbb{R}$ describe the performance of the model with the hyperparameter configuration $(\lambda, \gamma) \in \Theta$.

Naturally, the performance risk of a hyperparameter configuration $\theta \in \Theta$ is defined as the performance difference to the optimal performance,

$$d_\theta := f(\theta^*) - f(\theta). \tag{3.4}$$

The performance risk quantifies the risk of not tuning hyperparameters. The set $\{d_\theta \mid \theta \in \Theta\}$ describes an empirical distribution of performance differences over hyperparameter configurations.

We describe this empirical distribution using its mean d_{mean} and standard deviation d_{std},

$$d_{mean} = \frac{1}{|\Theta|} \sum_{\theta \in \Theta} d_\theta, \qquad d_{std} = \sqrt{\frac{\sum_{\theta \in \Theta} (d_\theta - d_{mean})^2}{|\Theta|}}. \tag{3.5}$$

Just like for the tuning risk introduced in (Weerts et al., 2020), we acknowledge that our assessment of the performance risk relies on the assumption that all units of risk hold equal significance. This assumption may be unrealistic in practice. For example, a performance risk of 0.01 might be considered more severe if the initial optimal performance is only 0.02, compared to the case where the optimal performance is 0.85. In the former scenario, the number of miss-classified instances is doubled if we consider the top-1 accuracy as performance measure.

To address this concern, we introduce the relative performance risk, defined as follows:

$$d_\theta^R := \frac{f(\theta^*) - f(\theta)}{f(\theta^*)}. \tag{3.6}$$

The resulting empirical distribution of the relative performance differences over hyperparameter configurations is described using its mean d_{mean}^R and its standard deviation d_{std}^R as in Equation 3.5.

In our analysis, we are not interested in the risk of the entirety of the hyperparameter performance space. Instead, we are specifically concerned with the performance risk of optimal hyperparameter configurations of the unpruned model. To achieve this, we define our region of interest (Θ) as the 0.5-ROI, which encompasses all hyperparameter configurations that achieve optimal performance within the top 0.5% in terms of top-1 accuarcy on the unpruned model. In this case, not tuning hyperparameters after pruning refers to training the pruned model with the optimal hyperparameter configuration from the unpruned model, $(\lambda, \gamma) \in$ 0.5-ROI.

There are several reasons behind our choice of $x = 0.5$ as a threshold for the optimal hyperparameters. Firstly, we assume that the initial hyperparameters of the basemodel have already been carefully tuned. If this is not the case, the performance risk of not tuning hyperparameters after pruning would likely be comparable to the general performance risk of not tuning hyperparameters at all, regardless of

the pruning process. By setting $x = 0.5$, we align with practical scenarios where a well-performing basemodel already exists, and the focus is on compressing the model to accommodate resource-constrained devices or deployment environments. In such cases, it is reasonable to assume that the basemodel has undergone hyperparameter tuning to achieve satisfactory performance.

Secondly, based on our initial visual analysis, we determined that the 0.5-ROI region covers a sufficiently large portion of the hyperparameter space on the basemodel. This region includes a substantial number of hyperparameter configurations that achieve performance within the top 0.5%, allowing for a comprehensive assessment of the performance risk.

Lastly, while the threshold $x = 0.5$ is a somewhat arbitrary choice, it provides a fair evaluation of the performance risk under the assumption that any proficient hyperparameter optimiser can successfully identify the best hyperparameter configurations within the top 0.5% for the basemodel.

In addition to numerically describing the performance risk and relative performance risk, we employ box plots to visually analyse the empirical distributions d_θ and d_θ^R. The box plots provide a concise summary of the data distribution by displaying key statistical measures such as the median, quartiles, and potential outliers. This visual analysis allows us to gain insights into the variation and distribution of the risk across various pruning ratios, methods (uniform and non-uniform), and models. For implementation and visualisation of the box plots we employ the graphing library Plotly (Inc., 2015).

Chapter 4

Results

In this chapter, we present our experimental results and discuss implications and findings. Unless otherwise indicated, we consider as performance metric the top-1 accuracy on a held-out validation dataset. We analyse and quantify the hyperparameter performance space to answer our initial research questions:

Q1 *What are the effects of structured pruning on the hyperparameter performance space of a model?*

Q2 *What is the performance risk when not tuning hyperparameters after pruning?*

As described in Chapter 3, ResNet-56 and MobileNetV2 are evaluated on the CIFAR-10 dataset and ResNet-50 on the ImageNet dataset. For all architectures, we approximate the hyperparameter performance space by spanning a grid around the prior values for learning rate and weight decay (cf. Table 3.4).

4.1 Effects of pruning on the hyperparameter performance space

The experimental results of MobileNetV2 and ResNet-56 on CIFAR-10 are presented in Figure 4.1 and Figure 4.2, respectively. For ResNet-50 on ImageNet, the effect of pruning on the performance landscape with respect to the top-1 and top-5 accuracy is displayed in Figure 4.3. These figures depict scatter and contour plots showcasing the relationship between weight decay, learning rate, and model performance across various pruning ratios and methods (uniform and non-uniform).

Rather than displaying the specific location of the single best hyperparameter configuration, our approach involves approximating the shape of the optimal configurations using confidence ellipses. This decision is driven by our focus on capturing the overall shape of the optimal hyperparameters rather than isolating the single best setting.

The figures display the 86.47% confidence ellipses fitted to the 0.5-ROI and 1.0-ROI of the respective pruned or unpruned model. In accordance with Section 3.3.1, the confidence ellipses depict the contour line at 2 standard deviations from the mean

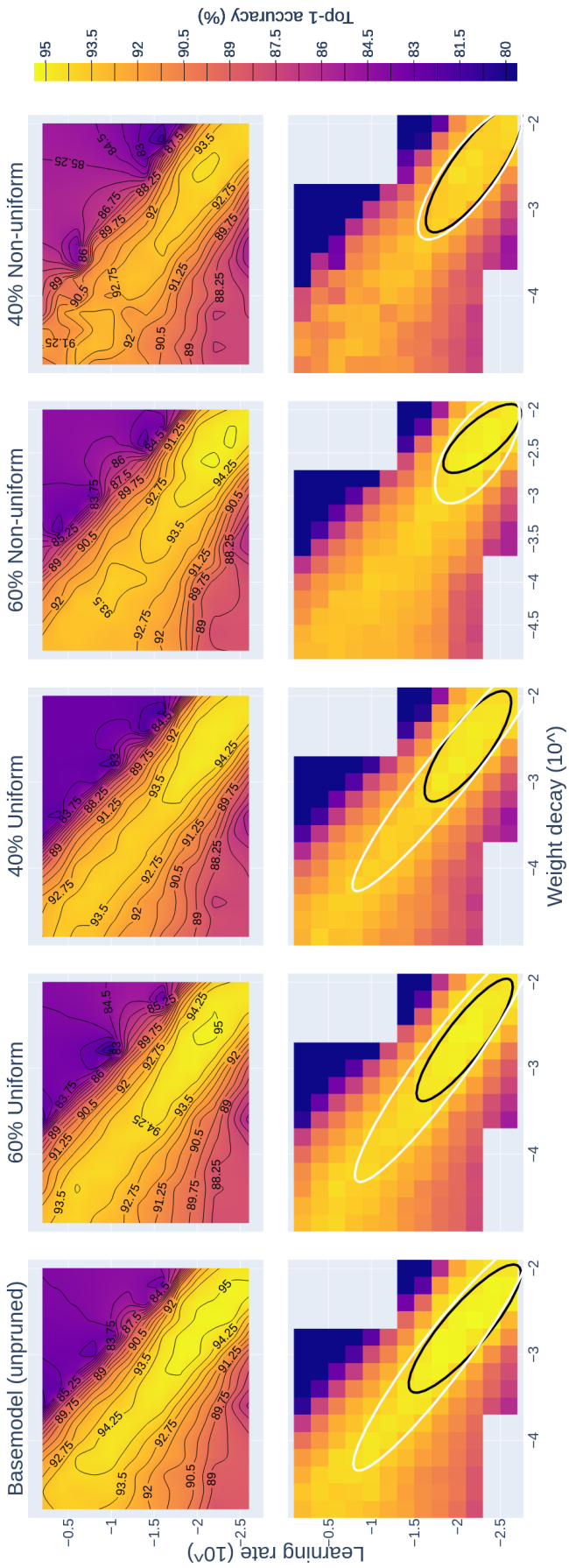

Figure 4.1: Hyperparameter performance space of MobileNetV2 under uniform and non-uniform magnitude pruning. Comparison of the 60% and 40% pruned architectures on CIFAR-10 with the unpruned MobileNetV2 (left). In the contour plots (top), points outside the colour bar range are removed and inter-/extrapolated, resulting in a visually smoother representation. The colour bar range captures the top 15% of observations across all MobileNetV2 architectures (unpruned and pruned). Confidence ellipses (bottom) are fitted to the hyperparameter configurations that achieve top-1 accuracy within the top 1% (white) and top 0.5% (black). Best viewed in colour.

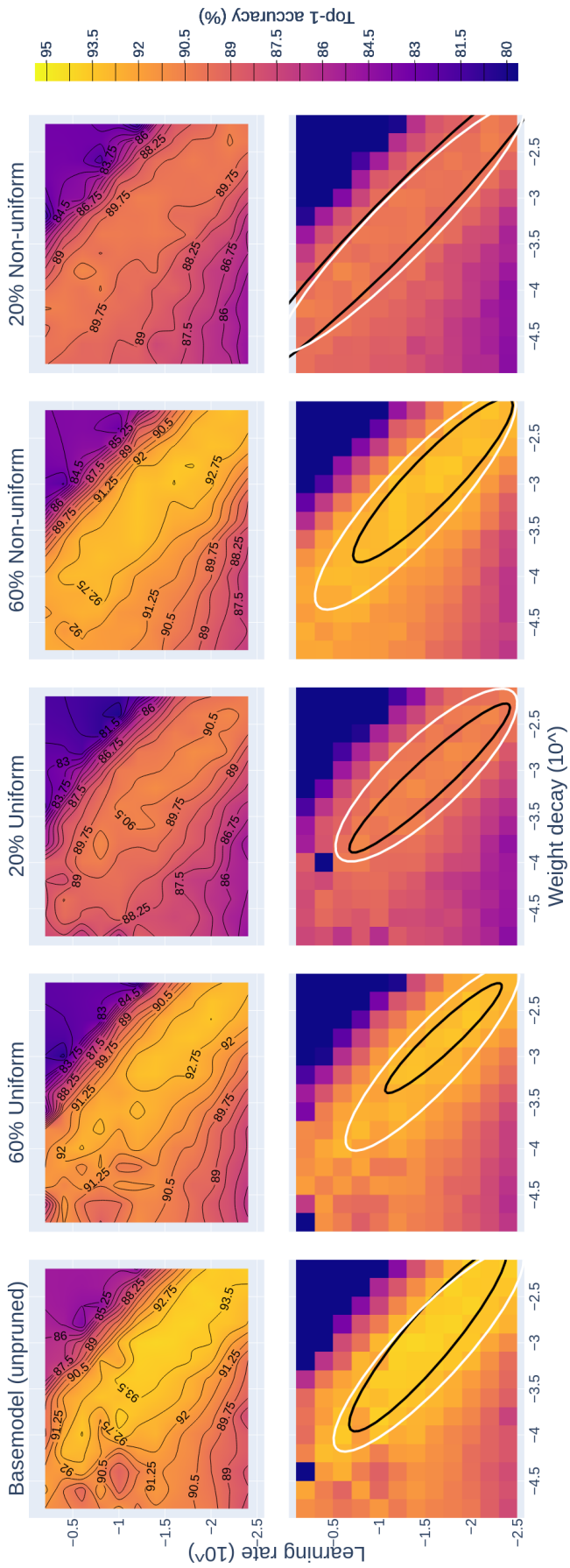

Figure 4.2: Hyperparameter performance space of ResNet-56 under uniform and non-uniform magnitude pruning. Comparison of the 60% and 20% pruned architectures on CIFAR-10 with the unpruned ResNet-56 (left). In the contour plots (top), points outside the colour bar range are removed and inter-/extrapolated, resulting in a visually smoother representation. The colour bar range captures the top 15% of observations across all ResNet-56 architectures (unpruned and pruned). Confidence ellipses (bottom) are fitted to the hyperparameter configurations that achieve top-1 accuracy within the top 1% (white) and top 0.5% (black). Best viewed in colour.

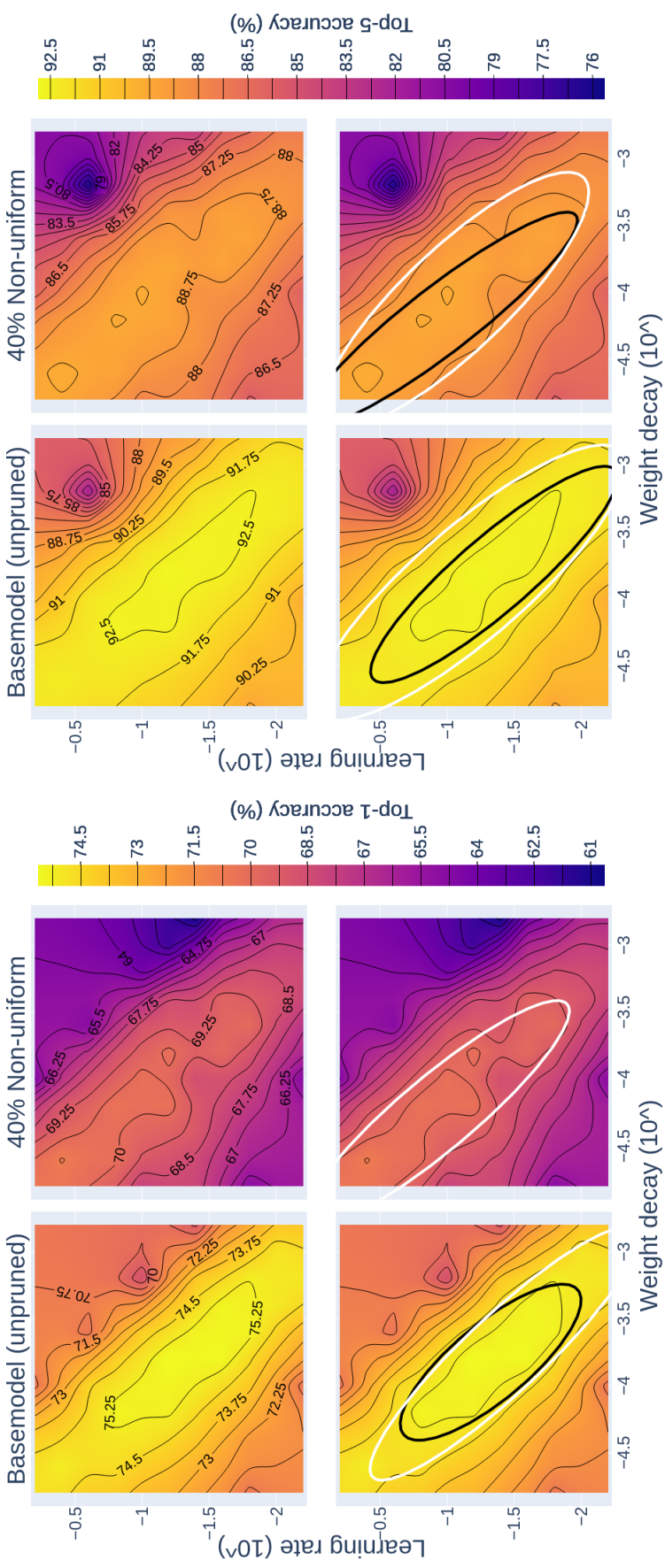

Figure 4.3: Hyperparameter performance space of ResNet-50 under non-uniform magnitude pruning. Comparison of the 40% pruned architectures on ImageNet with the unpruned ResNet-50. Confidence ellipses (bottom) are fitted to the hyperparameter configurations that achieve performance within the top 1% (white) and top 0.5% (black) given the top-1 accuracy (left) and top-5 accuracy (right). Best viewed in colour.

of the probability density function of the bivariate Gaussian fit to the 0.5-ROI and 1.0-ROI, respectively.

4.1.1 Visual analysis

Overall, the shape of the performance landscape appears to be quite consistent across all pruning strengths, pruning methods (uniform and non-uniform), datasets (CIFAR-10 and ImageNet), and model architectures (ResNet-56, MobileNetV2, and ResNet-50). In all plots an optimal band spanning from the top left (larger learning rate, smaller weight decay) to the bottom right (smaller learning rate, larger weight decay) is clearly evident. In this context, the overall shape is not solely defined by the absolute colour representation, meaning the absolute top-1 accuracy values. Instead, we refer to the variations and patterns observed in the data.

MobileNetV2 on CIFAR-10

For the unpruned MobileNetV2 on CIFAR-10 the visual analysis reveals that the optimal span of weight decay and learning rate values clearly extends from the bottom right to the top left. The best area appears closer to the bottom right, characterised by small learning rate values and larger weight decay values, as depicted by the contour line corresponding to a top-1 accuracy of 95%.

The baseline model achieves a best top-1 accuracy of 95.32%. For both uniform and non-uniform magnitude pruning on MobileNetV2 only a small reduction in performance compared to the basemodel is observed. Further, for both the uniform and non-uniform pruning method the optimal area and overall shape of the performance landscape remain consistent.

The best-observed top-1 accuracy of the 60% uniform magnitude-pruned model is 95.14%. For the 40% uniform magnitude-pruned architecture it is 94.94%. When considering non-uniform magnitude pruning, the 60% and 40% pruned models achieve a best top-1 accuracy of 95.18% and 94.48%, respectively. For the 40% non-uniform pruned model, it is worth noting that the optimal band at larger learning rate values is not as clearly defined. However, it remains distinctly evident for smaller learning rate values, as observed in the plots for the other pruning methods of MobileNetV2.

Across all pruning ratios and pruning methods (uniform and non-uniform) the optimal hyperparameter area remains in the bottom right, characterised by small learning rate and larger weight decay values. Despite a slight decrease in accuracy after pruning, the overall shape and optimal area remain consistent. This observation is clearly evident when considering the confidence ellipses fitted to 0.5-ROI, displayed as black ellipses in Figure 4.1.

However, considering the white ellipses corresponding to the 1.0-ROI, there is a small shift visible. For the basemodel and the uniformly pruned architectures these ellipses stretch across nearly the full optimal diagonal, indicating that hyperparameter configuration achieving optimal performance (within the top 1%) are widely spread. Yet, for non-uniform pruning this is not the case. For the non-uniform magnitude-pruned architecture the 1.0-ROI ellipses are of smaller size and are centred at the

bottom right, indicating that the optimal hyperparameter configurations are not as widely spread.

While there is a difference in the shape of the 1.0-ROI confidence ellipse between the uniform and non-uniform pruned architectures, we point out that we do not capture the entirety of the optimal hyperparameter configurations in the data, potentially affecting the shape of the confidence ellipses. With that said, it is important to interpret the confidence ellipses as a representation of the available data and the observed trends, but they may not capture every possible optimal configuration. Future studies with a larger and more comprehensive dataset could provide further insights into the shape and distribution of the optimal (weight decayand learning rate) configurations after pruning.

ResNet-56 on CIFAR-10

For ResNet-56 on CIFAR-10 similar observations can be made compared to MobileNetV2 on the same dataset. While the optimal band spanning from the bottom right to the top left is visible across all pruning ratios and methods (uniform and non-uniform), it is not as clearly defined as for MobileNetV2. However, the shape of the performance landscape remains consistent across different pruning methods for ResNet-56, just as it was the case for MobileNetV2. We conclude that the overall shape of the performance landscape remains consistent across different pruning methods and strengths but not across different architecture families.

The unpruned ResNet-56 achieves a best top-1 accuracy of 94.05%. However, there is a noticeable drop in performance after pruning. The uniformly pruned model achieves a top-1 accuracy of up to 93.52% and 91.08% for pruning ratios of 60% and 20%, respectively. Similarly, the 60% and 20% non-uniform pruned architectures achieve top-1 accuracies of 93.55% and 90.98%, respectively. As for the results observed on MobileNetV2, the drop in performance is comparable between the uniform and non-uniform magnitude pruning methods.

Further, the location of the optimal hyperparameters remains consistent across different pruning ratios and methods, as shown in Figure 4.2. This observation is supported by the 86.47% confidence ellipses fitted to the 0.5-ROI and 1.0-ROI as their position and shape remain mostly consistent. This indicates that there is no significant shift in the optimal hyperparameters after pruning.

However, there is one notable outlier: the 20% non-uniform magnitude-pruned ResNet-56. Unlike the observed trend in the non-uniform pruned MobileNetV2, where the 1.0-ROI confidence ellipses were smaller compared to the basemodel and uniform pruned architectures, we observe the inverse effect for the 20% non-uniform pruned ResNet-56. In this case, both the 1.0-ROI and 0.5-ROI confidence ellipses become larger and stretch across the entirety of the optimal hyperparameter region, as depicted in Figure 4.2.

The inconsistency of the observed trend regarding the impact of pruning on the hyperparameter performance space and on the optimal hyperparameters suggests that its effects can vary depending on the specific architecture and pruning method

used. Different architectures have unique characteristics and behaviours, and the pruning algorithms may interact with these characteristics in different ways.

ResNet-50 on ImageNet

Figure 4.4 illustrates the grid used for training and evaluating ResNet-50 on the ImageNet dataset. Similar to the experimental results on CIFAR-10, the optimal hyperparameters span from smaller learning rate values with larger weight decay values to larger learning rate values with smaller weight decay values. This optimal band is also observed for the top-5 accuracy as performance metric. More generally, the performance landscape for the top-1 and top-5 accuracy as performance metrics are very similar for both the unpruned and pruned models.

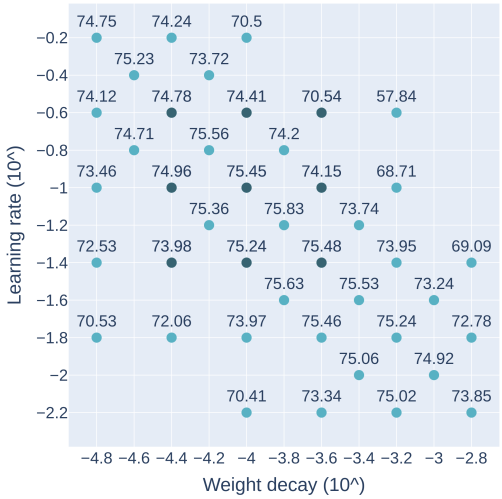

Figure 4.4: The weight decay and learning rate grid for ResNet-50 experiments on ImageNet. The darker coloured 3×3 grid shows the initial grid around the prior which we extended. Annotations show the top-1 accuracy of the unpruned ResNet-50 trained with the specific hyperparameters.

The unpruned ResNet-50 achieves a maximum top-1 accuracy of 75.83%, accompanied by a top-5 accuracy of 92.84%. Notably, this top-5 accuracy also represents the best performance achieved across all weight decay and learning rate configurations for ResNet-50. For the non-uniform magnitude-pruned model, the best top-1 accuracy obtained is 70.81%, showcasing a significant decrease in performance. Similarly, the best top-5 accuracy achieved is 89.73% with the same hyperparameter configuration. These results demonstrate the trade-off between model size reduction through pruning and the resulting performance decline.

The confidence ellipses visualised in Figure 4.3 show a similar shape. The visualisation of the 40% non-uniform magnitude pruned ResNet-50 architecture exclusively displays the 1.0-ROI ellipse. This choice is attributed to the fact that fewer than five hyperparameter configurations achieve a top-1 accuracy within the top 0.5%. We require a minimum of five data points for fitting the confidence ellipse to the specific ROI since this minimum number guarantees an unique solution.

Once again, similar to the experiments on CIFAR-10, there is no significant shift in the optimal hyperparameters after pruning. In the case of the unpruned model, the optimal hyperparameters are scattered across the optimal band, forming a larger, centrally located cluster. This wide dispersion is visually represented by the longer lengths of the 0.5-ROI and 1.0-ROI ellipses. Likewise, for the pruned model, the optimal regions are spread diagonally, as illustrated in Figure 4.3 and as indicated by the length of the 1.0-ROI ellipse.

Relationship between weight decay and learning rate

Our experimental results clearly demonstrate that a larger learning rate necessitates a smaller weight decay, and conversely, a smaller learning rate requires a larger weight decay for optimal performance. This empirical evidence supports the weight decay and learning rate relationship as previously presented by L. N. Smith (2018). Our findings align with the theoretical understanding that learning rate and weight decay tend to have inverse effects on the weight update of a model (with SGD) as demonstrated in Equation 2.1.

4.1.2 Interpretation of confidence ellipses

Through the visual analysis of contour and scatter plots, accompanied by the visualisation of confidence ellipses, we have gained an initial understanding that there is no significant shift in the optimal hyperparameters. This suggests that the impact of pruning on the hyperparameter performance space with respect to the weight decay and learning rate is insignificant. However, to further support this initial hypothesis and gain a deeper understanding, we will now provide a detailed analysis of the confidence ellipses, as they effectively model the area encompassing the optimal hyperparameters.

Table 4.1 displays the length, width, and centre of the confidence ellipses fitted to the 1.0-ROI and 0.5-ROI for the pruned and unpruned architectures of ResNet-56, MobileNetV2, and ResNet-50. In this analysis, the length represents the longest diameter of the ellipse, while the width represents its shortest diameter. Our primary focus is to compare the confidence ellipses of the pruned models with the respective unpruned model, aiming to determine whether there is a noticeable shift in the hyperparameter space after pruning.

MobileNetV2 on CIFAR-10 The numerical representation of the 1.0-ROI and 0.5-ROI confidence ellipses for MobileNetV2 support our visual interpretations. The difference of the 1.0-ROI confidence ellipses between the uniform and non-uniform magnitude-pruned models is clearly evident as determined by the different lengths of the ellipses. However, both 1.0-ROI and 0.5-ROI ellipses show similar centres across all pruning methods, indicating that, on average, there is no significant shift of the optimal hyperparameters after pruning.

ResNet-56 on CIFAR-10 Across all pruning ratios, including the unpruned model, the centre of the ellipses fitted to the 1.0-ROI or 0.5-ROI remain consistent. There is a negligible maximum difference of (0.2, 0.3) for the 1.0-ROI and (0.3, 0.3) for the 0.5-ROI in the (weight decay, learning rate) domain. This supports the initial observation, that there is no significant shift of the optimal hyperparameters.

Further, the width and length of all ellipses fitted to 1.0-ROI demonstrate consistency, with a slight fluctuation of up to +/- 0.07 for the width and up to +/- 0.76 for the length, relative to the ellipse of the basemodel. In terms of the 0.5-ROI, the width and length of the ellipses display more variation, without a clear discernible trend.

	1.0-ROI			0.5-ROI		
	length	width	centre	length	width	centre
MobileNetV2 on CIFAR-10						
Basemodel	3.32	0.56	$(-3.0, -1.8)$	1.91	0.51	$(-2.7, -2.1)$
60% Uniform	3.37	0.57	$(-3.0, -1.8)$	1.75	0.45	$(-2.7, -2.1)$
40% Uniform	3.55	0.54	$(-2.9, -1.9)$	1.55	0.50	$(-2.6, -2.1)$
60% Non-uniform	1.46	0.66	$(-2.5, -2.2)$	1.13	0.43	$(-2.3, -2.3)$
40% Non-uniform	1.86	0.59	$(-2.6, -2.2)$	1.78	0.54	$(-2.5, -2.2)$
ResNet-56 on CIFAR-10						
Basemodel	2.93	0.73	$(-3.1, -1.5)$	2.58	0.52	$(-3.0, -1.5)$
60% Uniform	2.69	0.66	$(-3.0, -1.6)$	1.71	0.29	$(-2.8, -1.7)$
20% Uniform	2.60	0.79	$(-3.1, -1.5)$	2.34	0.4	$(-3.1, -1.6)$
60% Non-uniform	3.12	0.78	$(-3.2, -1.4)$	2.43	0.52	$(-3.0, -1.6)$
20% Non-uniform	3.74	0.66	$(-3.3, -1.2)$	4.76	0.58	$(-3.3, -1.2)$
ResNet-50 on ImageNet						
Basemodel	2.69	0.58	$(-3.8, -1.4)$	1.69	0.57	$(-3.8, -1.3)$
40% Non-uniform	2.53	0.55	$(-4.3, -0.9)$	–	–	–

Table 4.1: Width, height, and centre of confidence ellipses fitted to the 1.0-ROI and 0.5-ROI of ResNet-56, MobileNetV2, and ResNet-50. Specifically, ellipses are fit to x-ROI if and only if x-ROI contains at least 5 elements. The centre is denoted in the (weight decay, learning rate) domain. Further, the width denotes the shorter side of the ellipse and the length the longer side. All ellipses are fitted in the log-scale. See also Figure 4.1, Figure 4.2, and Figure 4.3 for visualisations of the confidence ellipses. x% Uniform or x% Non-uniform denotes uniform or non-uniform magnitude pruning with a pruning ratio of x%.

ResNet-50 on ImageNet For ResNet-50, the analysis is somewhat constrained due to the limited number of data points. However, it is noteworthy that the observations derived from the experiments on CIFAR-10 are applicable to the ResNet-50 architecture as well. Specifically, the centre of the confidence ellipses for ResNet-50 remain consistent after pruning, with a maximum difference to the basemodel's confidence ellipses of 0.3 observed in both the weight decay and learning rate dimensions. Furthermore, similar to the findings for MobileNetV2 and ResNet-56, the width and length of the ellipses for ResNet-50 show negligible variations.

The analysis of the confidence ellipses aligns with our findings from the visual analysis. We found no significant evidence that pruning has a substantial impact on the hyperparameter performance space in the form of, for example, a clear shift or alteration of the optimal hyperparameters. However, our analysis does reveal variations in the optimal hyperparameters as indicated, for example, by the fluctuating length of the confidence ellipses. In the next section, we will delve deeper into understanding how these fluctuations affect the final accuracy and evaluate the potential risk of not tuning hyperparameters after pruning.

4.2 Performance risk of not tuning hyperparameters after pruning

The mean and standard deviation of both the performance risk and relative performance risk are presented in Table 4.2. We refer to Section 3.3.2 for definitions and underlying methodology. The mean and standard deviation provide insights into the average and variability of the risk across different model architectures and pruning methods. In general, a higher absolute performance risk indicates a larger deviation from the optimal performance, highlighting the potential loss in performance when hyperparameters are not tuned after pruning.

	d_{mean}	d_{std}	d^R_{mean}	d^R_{std}
MobileNetV2 on CIFAR-10				
Basemodel	0.219	0.128	0.002	0.001
60% Uniform	0.373	0.266	0.004	0.003
40% Uniform	0.413	0.284	0.004	0.003
60% Non-uniform	0.591	0.391	0.006	0.004
40% Non-uniform	0.568	0.281	0.006	0.003
ResNet-56 on CIFAR-10				
Basemodel	0.271	0.137	0.003	0.001
60% Uniform	0.622	0.428	0.007	0.005
20% Uniform	0.623	0.325	0.007	0.004
60% Non-uniform	0.436	0.256	0.005	0.003
20% Non-uniform	0.890	0.258	0.010	0.003
ResNet-50 on ImageNet				
Basemodel	0.293	0.142	0.004	0.002
40% Non-uniform	0.864	0.467	0.012	0.007

Table 4.2: Mean and standard deviation of the performance risk (d_{mean}, d_{std}) and relative performance risk (d^R_{mean}, d^R_{std}) when not tuning hyperparameters after pruning.

In addition to the numerical representations of the risk using its mean and standard deviation, Figure 4.5 presents box plots showcasing the distribution of the performance risk and relative performance risk, respectively. These visualisations provide insights into the locality, spread, and skewness of the risk across different architectures and pruning methods.

Overall, the risk increases with pruning. In Figure 4.5, the darker coloured box plots describe the risk on the basemodel. Given that we only consider the 0.5-ROI, the absolute performance risk on the unpruned model is at most 0.5% since 0.5-ROI describes the set of hyperparameter configuration achieving a top-1 accuracy within the top 0.5%. Across all pruning methods and model architectures the absolute performance risk increases after pruning as indicated by the higher median and mean compared to the risk on the unpruned model.

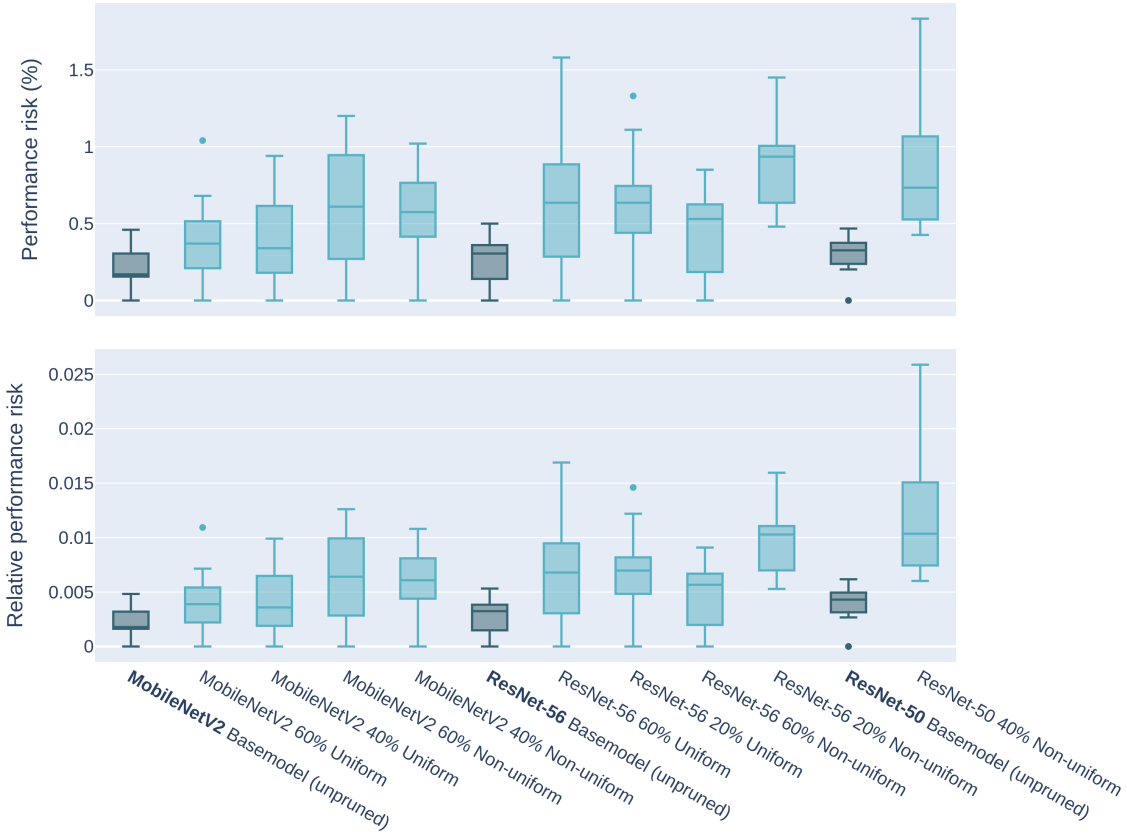

Figure 4.5: Absolute and relative performance risk given 0.5-ROI of the basemodel (unpruned) for the pruned and unpruned MobileNetV2, ResNet-56, and ResNet-50 architectures, respectively. The performance risk is measured given the best achieved top-1 accuracy of the respective architecture (cf. Section 3.3). Darker coloured box plots indicate the risk of the basemodel with respect to the best top-1 accuracy for that model.

This observation indicates that the optimal hyperparameters of the pruned model differ from those of the unpruned model. If this wouldn't be the case we would expect the risk to remain relatively stable and comparable to the risk of the basemodel. This observation aligns with the slight fluctuation of the performance landscape discussed in Section 4.1.

Examining the absolute performance risk, we observe a trend for MobileNetV2. For MobileNetV2 uniform magnitude-pruned models exhibit lower mean performance risks (d_{mean}) of 0.373 and 0.413 compared to non-uniform pruned models with mean performance risks of 0.591 and 0.568, as displayed in Table 4.2. Similarly, when considering the centre of the data displayed in Figure 4.5, both non-uniform pruned architectures have a higher median compared to the uniform magnitude-pruned MobileNetV2.

However, for ResNet-56 this observation of a trend does not hold. For the ResNet-56 architecture, the 20% non-uniform magnitude-pruned configuration exhibits the highest average performance risk, as indicated by a mean performance risk of 0.89.

This means that there is a drop in performance of 0.89 on average when training the pruned architecture with the optimal hyperparameters from the 0.5-ROI region of the basemodel. Further, the 60% non-uniform magnitude-pruned architecture exhibits the lowest average performance risk for ResNet-56 of 0.436. Thus, for the non-uniform pruned architectures of ResNet-56 we observe both the highest and lowest average performance risk. Similarly, the median displayed in the box plots for ResNet-56 does not seem to follow a specific pattern or trend (cf. Figure 4.5).

Intuitively, one would expect a consistently higher risk for one pruning method (uniform or non-uniform) as this would indicate a relation between performance risk and pruning method. While for MobileNetV2 non-uniform pruning shows a greater performance risk on average compared to uniform pruning, this observation does not hold for ResNet-56, where there is no clear difference in performance risk between uniform and non-uniform pruning methods.

Similarly, across pruning ratios (60% and 40% for MobileNet or 60% and 20% for ResNet-56) there is not one pruning ratio consistently achieving higher risk compared to the other one. Intuitively, one would expect an increasing risk with a lower pruning ratio if there is a correlation between performance risk and pruning ratio.

In addition to the absolute performance risk, the relative performance risk allows for comparing the risk across different models and datasets. The dispersion of the relative performance risk, as denoted by the standard deviation d^R_{std}, ranges from 0.003 to 0.007 for the pruned models. Similarly, the mean values of the relative performance risk range from 0.004 for the non-uniform pruned MobileNetV2 to 0.012 for the 40% non-uniform pruned ResNet-50. The narrow range of values indicates a consistent dispersion of risk across the evaluated scenarios.

Based on the analysis of the performance risk distribution visualised in Figure 4.5, we observe that the risk remains within a range of 2× the risk of the basemodel for most optimal hyperparameter configurations. Specifically, for MobileNetV2 and ResNet-56, the relative performance risk of the basemodel is approximately 0.005, and for the majority of pruning techniques, about 75% of the configurations on 0.5-ROI exhibit a relative performance risk of 0.01 or below. This is indicated by the upper border of the box on the box plots, which represents the third quartile. Similarly, for ResNet-50, more than 50% of the optimal hyperparameter configurations of the basemodel remain within a range of 2× the relative performance risk of the basemodel. These observations highlight that the performance risk, while slightly increasing after pruning, generally remains within acceptable bounds for the majority of hyperparameter configurations.

The observation that the performance risk remains within an acceptable range after pruning has practical implications for the deployment of pruned models. Specifically, our findings suggest that the pruned models can maintain satisfactory performance levels without the need for extensive hyperparameter adjustments.

In summary, although the performance risk increases after pruning, it generally remains within acceptable bounds for the majority of hyperparameter configurations. This lack of a significant increase of risk after pruning can be attributed to the

absence of a notable shift or trend in the performance landscape after pruning, as discussed in Section 4.1. Additionally, we observed no clear pattern or shift in risk across different pruning ratios or methods, suggesting that the relationship between performance risk and pruning is not strictly defined.

Chapter 5

Conclusions

In this thesis, we examined the impact of uniform and non-uniform structured magnitude pruning on the hyperparameter performance space of a model. In an empirical study using widely-adopted benchmark datasets and image classification models in computer vision, we addressed two open questions in the research community:

Q1 *What are the effects of structured pruning on the hyperparameter performance space of a model?*

Q2 *What is the performance risk when not tuning hyperparameters after pruning?*

Q1: Effects of pruning For the first question, we approximated the performance landscape by spanning a grid of hyperparameter configurations around an initial prior configuration. Through visual and statistical analysis, we compared the resulting performance landscapes of both the unpruned and pruned models to evaluate the impact of pruning on the model's hyperparameters. Surprisingly, we did not observe a significant impact of pruning on the performance landscape of a model. This observation was consistent across all tested pruning methods, as evidenced by the similar shape observed in the performance landscapes. This indicates that the performance landscape remains relatively stable across different structured pruning methods. Moreover, we observed no clear trend or pattern regarding a shift or change in the optimal hyperparameters when employing different methods (uniform and non-uniform) or pruning strengths.

Q2: Performance risk For the second question, we quantified the potential loss in performance associated with not tuning hyperparameters after pruning. Our analysis revealed that there is no significant relation between the pruning ratio or method (uniform or non-uniform) and the performance risk when hyperparameters are not tuned after pruning. However, while our data validated an increased performance risk after pruning, showing an instability of the optimal hyperparameters, this increased risk remains within acceptable bounds for most configurations.

Based on the combined findings from our two research questions, we derive practical recommendations. Our results suggest that the amount of hyperparameter tuning,

specifically for the learning rate and weight decay, can be minimised after pruning. This is due to two key observations: Firstly, we found that the performance risk, which measures the potential decrease in performance after pruning, remains within an acceptable range for the majority of hyperparameter configurations. Secondly, our analysis of the performance landscape revealed that pruning does not have a substantial impact on the overall shape and structure of the landscape. This implies that the optimal hyperparameter configuration of the basemodel can serve as a reasonable starting point for the pruned model.

Contribution Our research contributes to the current state of the literature by improving our understanding of hyperparameters in the context of pruning. Moreover, our results can serve as practical guidelines for practitioners in the field interested in maximising their model's performance after pruning. Further, our work provides empirical evidence supporting the previously presented relationship between weight decay and learning rate, as described by S. L. Smith et al. (2018). Across all model architectures and datasets, our experiments clearly demonstrate the inverse relationship between the optimal learning rate and weight decay values in the form of an optimal band spanning across the landscape. Furthermore, our study aligns with the existing literature on hyperparameter transferability across network width scaling (Park et al., 2019; Yang et al., 2022). The consistent stability of the performance landscape under pruning, as demonstrated by our results, can inform the development of hyperparameter optimisation techniques.

Our results raise interesting new questions relating to the factors that do affect the weight decay and learning rate relation and, conversely, the hyperparameters that are affected by pruning.

5.1 Limitations and future work

Our study has some limitations that could be addressed in future research: Firstly, the study focuses solely on convolutional neural networks for image classification tasks. To provide a broader understanding of the impact of pruning on different types of models, it would be valuable to extend our work to other domains, such as language models in NLP. Secondly, we only explore magnitude-based pruning methods. Considering the variety of pruning techniques available, further research could investigate the effects of other pruning methods and techniques on the hyperparameter performance space.

In this context, the chosen pruning setup, particularly the specific pruning pipeline of retraining from scratch after pruning, poses a limitation of this work. As visualised in Figure 1.1, a number of different pruning pipelines are available, each with their own characteristics and potential impacts on the optimal hyperparameters. Exploring different pruning setups and evaluating their influence on hyperparameters would provide deeper insights into the impact of pruning on the hyperparameters of a model.

Lastly, our study focuses primarily on the learning rate and weight decay as hyperparameters. However, a model's performance is influenced by a wide range of hyperparameters, including the number of epochs, batch size, and the learning rate scheduler, among others. Future work could assess the impact of pruning on this wider range of hyperparameters to gain a better understanding of their interactions and the implications for model performance.

5.2 Ethical consideration

> "Computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it" (Redmon and Farhadi (2018), p. 4)

Compression of deep learning models for deployment on resource-constrained devices enables the use of machine learning in many areas that were previously not feasible. However, as with any technology, there is also the risk of misusing this opportunity. We outline ethical considerations in deep learning compression and computer vision as our study aims to contribute to these fields.

One potential misuse of compressed deep learning models is for malicious purposes, such as unauthorised surveillance, invasions of privacy, or cyber-attacks. For example, a compressed deep learning model running on a small camera could be used to identify and track individuals without their knowledge or consent, leading to privacy violations. To mitigate these risks, it is important to develop and implement ethical guidelines and regulations for the deployment of deep learning models in edge devices.

Pruning of deep learning models often results in a slight reduction in model accuracy and performance. This can be particularly concerning in applications where accuracy is critical, such as in the medical field or security-related applications. For example, a low-accuracy model can lead to significant harm by a misdiagnosis of a medical condition. Therefore, it is crucial to carefully evaluate the trade-off between model accuracy and compression in these applications. By getting a better understanding of the influence of pruning on the hyperparameter space of a model, we help to mitigate this issue as model hyperparameters are tightly coupled to the performance of a model.

Bibliography

Anwar, S., Hwang, K., & Sung, W. (2015). Structured Pruning of Deep Convolutional Neural Networks [arXiv:1512.08571 [cs, stat]]. https://doi.org/10.48550/arXiv.1512.08571

Arora, S., Cohen, N., & Hazan, E. (2018). On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization [arXiv:1802.06509 [cs]]. https://doi.org/10.48550/arXiv.1802.06509

Ayi, M., & El-Sharkawy, M. (2020). RMNv2: Reduced Mobilenet V2 for CIFAR10. *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 0287–0292. https://doi.org/10.1109/CCWC47524.2020.9031131

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [arXiv:1308.3432 [cs]]. https://doi.org/10.48550/arXiv.1308.3432

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(2).

Blalock, D., Ortiz, J. J. G., Frankle, J., & Guttag, J. (2020). What is the State of Neural Network Pruning? [arXiv:2003.03033 [cs, stat]]. https://doi.org/10.48550/arXiv.2003.03033

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners [arXiv:2005.14165 [cs]]. https://doi.org/10.48550/arXiv.2005.14165

Cai, Y., Hua, W., Chen, H., Suh, G. E., De Sa, C., & Zhang, Z. (2022). Structured Pruning is All You Need for Pruning CNNs at Initialization [arXiv:2203.02549 [cs]]. https://doi.org/10.48550/arXiv.2203.02549

Chang, X., Li, Y., Oymak, S., & Thrampoulidis, C. (2020). Provable Benefits of Overparameterization in Model Compression: From Double Descent to Pruning Neural Networks [arXiv:2012.08749 [cs, stat]]. https://doi.org/10.48550/arXiv.2012.08749

Chen, H. (2018). A pytorch implement of mobileNet v2 on cifar10. Retrieved May 28, 2023, from https://github.com/chenhang98/mobileNet-v2_cifar10

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2020). A Survey of Model Compression and Acceleration for Deep Neural Networks [arXiv:1710.09282 [cs]]. https://doi.org/10.48550/arXiv.1710.09282

Child, R. (2021). Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images [arXiv:2011.10650 [cs]]. https://doi.org/10.48550/arXiv.2011.10650

Choudhary, T., Mishra, V., Goswami, A., & Sarangapani, J. (2020). A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, *53*(7), 5113–5155. https://doi.org/10.1007/s10462-020-09816-7

Crowley, E. J., Turner, J., Storkey, A., & O'Boyle, M. (2019). A Closer Look at Structured Pruning for Neural Network Compression [arXiv:1810.04622 [cs, stat]]. https://doi.org/10.48550/arXiv.1810.04622

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv:1810.04805 [cs]]. https://doi.org/10.48550/arXiv.1810.04805

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [arXiv:2010.11929 [cs]]. https://doi.org/10.48550/arXiv.2010.11929

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*(61), 2121–2159. Retrieved April 29, 2023, from http://jmlr.org/papers/v12/duchi11a.html

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural Architecture Search: A Survey [arXiv:1808.05377 [cs, stat]]. https://doi.org/10.48550/arXiv.1808.05377

Embedl Model Optimization SDK (Version 2023.4.0+torch). (2023). Embedl AB. https://www.embedl.com/

Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., & Joulin, A. (2021). Training with Quantization Noise for Extreme Model Compression [arXiv:2004.07320 [cs, stat]]. https://doi.org/10.48550/arXiv.2004.07320

Frankle, J., & Carbin, M. (2019). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [arXiv:1803.03635 [cs]]. https://doi.org/10.48550/arXiv.1803.03635

Gale, T., Elsen, E., & Hooker, S. (2019). The State of Sparsity in Deep Neural Networks [arXiv:1902.09574 [cs, stat]]. https://doi.org/10.48550/arXiv.1902.09574

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks [ISSN: 1938-7228]. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256. Retrieved May 1, 2023, from https://proceedings.mlr.press/v9/glorot10a.html

Guo, Y., Yao, A., & Chen, Y. (2016). Dynamic Network Surgery for Efficient DNNs [arXiv:1608.04493 [cs]]. https://doi.org/10.48550/arXiv.1608.04493

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both Weights and Connections for Efficient Neural Networks [arXiv:1506.02626 [cs]]. https://doi.org/10.48550/arXiv.1506.02626

Hassibi, B., G.Stork, D., & Wolff, G. (1993). Optimal Brain Surgeon and general network pruning, 293–299 vol.1. https://doi.org/10.1109/ICNN.1993.298572

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification [arXiv:1502.01852 [cs]]. https://doi.org/10.48550/arXiv.1502.01852

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Deep Residual Learning for Image Recognition [arXiv:1512.03385 [cs]]. https://doi.org/10.48550/arXiv.1512.03385

He, Y., Kang, G., Dong, X., Fu, Y., & Yang, Y. (2018). Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks [arXiv:1808.06866 [cs]]. https://doi.org/10.48550/arXiv.1808.06866

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for MobileNetV3 [arXiv:1905.02244 [cs]]. https://doi.org/10.48550/arXiv.1905.02244

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [arXiv:1704.04861 [cs]]. https://doi.org/10.48550/arXiv.1704.04861

Idelbayev, Y. (2018). Proper ResNet Implementation for CIFAR10/CIFAR100 in PyTorch. Retrieved May 6, 2023, from https://github.com/akamaster/pytorch_resnet_cifar10

Inc., P. T. (2015). *Collaborative data science*. https://plot.ly

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, *13*(4), 455–492. https://doi.org/10.1023/A:1008306431147

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization [arXiv:1412.6980 [cs]]. https://doi.org/10.48550/arXiv.1412.6980

Kohavi, R., & John, G. H. (1995). Automatic Parameter Selection by Minimizing Estimated Error. In A. Prieditis & S. Russell (Eds.), *Machine Learning Proceedings 1995* (pp. 304–312). Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-377-6.50045-1

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

kuangliu. (2017). Train CIFAR10 with PyTorch. Retrieved May 28, 2023, from https://github.com/kuangliu/pytorch-cifar

Lecun, Y., Denker, J., & Solla, S. (1989). Optimal Brain Damage. *2*, 598–605.

Lee, N., Ajanthan, T., & Torr, P. H. S. (2019). SNIP: Single-shot Network Pruning based on Connection Sensitivity [arXiv:1810.02340 [cs]]. https://doi.org/10.48550/arXiv.1810.02340

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2017). Pruning Filters for Efficient ConvNets [arXiv:1608.08710 [cs]]. https://doi.org/10.48550/arXiv.1608.08710

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization [arXiv:1603.06560 [cs, stat]]. https://doi.org/10.48550/arXiv.1603.06560

Li, Y., Zhao, P., Yuan, G., Lin, X., Wang, Y., & Chen, X. (2022). Pruning-as-Search: Efficient Neural Architecture Search via Channel Pruning and Structural

Reparameterization [arXiv:2206.01198 [cs]]. https://doi.org/10.48550/arXiv.2206.01198

Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and Quantization for Deep Neural Network Acceleration: A Survey [arXiv:2101.09671 [cs]]. https://doi.org/10.48550/arXiv.2101.09671

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). Microsoft COCO: Common Objects in Context [arXiv:1405.0312 [cs]]. https://doi.org/10.48550/arXiv.1405.0312

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning Efficient Convolutional Networks through Network Slimming [arXiv:1708.06519 [cs]]. https://doi.org/10.48550/arXiv.1708.06519

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2019). Rethinking the Value of Network Pruning [arXiv:1810.05270 [cs, stat]]. https://doi.org/10.48550/arXiv.1810.05270

Ma, X., Lin, S., Ye, S., He, Z., Zhang, L., Yuan, G., Tan, S. H., Li, Z., Fan, D., Qian, X., Lin, X., Ma, K., & Wang, Y. (2020). Non-Structured DNN Weight Pruning – Is It Beneficial in Any Platform? [arXiv:1907.02124 [cs, stat]]. https://doi.org/10.48550/arXiv.1907.02124

maintainers, T., & contributors. (2016). *Torchvision: Pytorch's computer vision library*. GitHub. Retrieved May 6, 2023, from https://github.com/pytorch/vision

Mockus, J. (1975). On the Bayes Methods for Seeking the Extremal Point. *IFAC Proceedings Volumes*, *8*(1, Part 1), 428–431. https://doi.org/10.1016/S1474-6670(17)67769-3

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2017). Pruning Convolutional Neural Networks for Resource Efficient Inference [arXiv:1611.06440 [cs, stat]]. https://doi.org/10.48550/arXiv.1611.06440

Neill, J. O. (2020). An Overview of Neural Network Compression [arXiv:2006.03669 [cs, stat]]. https://doi.org/10.48550/arXiv.2006.03669

Neyman, J. (1937). Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *236*(767), 333–380. https://doi.org/10.1098/rsta.1937.0005

Novello, P., Poëtte, G., Lugato, D., & Congedo, P. M. (2022). Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning [arXiv:2207.06216 [cs, stat]]. https://doi.org/10.48550/arXiv.2207.06216

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature Visualization. *Distill*, *2*(11), e7. https://doi.org/10.23915/distill.00007

OpenAI. (2023). GPT-4 Technical Report [arXiv:2303.08774 [cs]]. https://doi.org/10.48550/arXiv.2303.08774

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., … Zhang, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning [arXiv:1912.06680 [cs, stat]]. https://doi.org/10.48550/arXiv.1912.06680

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks [arXiv:1511.08458 [cs]]. https://doi.org/10.48550/arXiv.1511.08458

Park, D. S., Sohl-Dickstein, J., Le, Q. V., & Smith, S. L. (2019). The Effect of Network Width on Stochastic Gradient Descent and Generalization: An Empirical Study [arXiv:1905.03776 [cs, stat]]. https://doi.org/10.48550/arXiv.1905.03776

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library [arXiv:1912.01703 [cs, stat]]. https://doi.org/10.48550/arXiv.1912.01703

Probst, P., Bischl, B., & Boulesteix, A.-L. (2018). Tunability: Importance of Hyperparameters of Machine Learning Algorithms [arXiv:1802.09596 [stat]]. https://doi.org/10.48550/arXiv.1802.09596

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement [arXiv:1804.02767 [cs]]. https://doi.org/10.48550/arXiv.1804.02767

Renda, A., Frankle, J., & Carbin, M. (2020). Comparing Rewinding and Fine-tuning in Neural Network Pruning [arXiv:2003.02389 [cs, stat]]. https://doi.org/10.48550/arXiv.2003.02389

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge [arXiv:1409.0575 [cs]]. https://doi.org/10.48550/arXiv.1409.0575

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). MobileNetV2: Inverted Residuals and Linear Bottlenecks [arXiv:1801.04381 [cs]]. https://doi.org/10.48550/arXiv.1801.04381

Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M., & Villalobos, P. (2022). Compute Trends Across Three Eras of Machine Learning [arXiv:2202.05924 [cs]]. https://doi.org/10.48550/arXiv.2202.05924

Shala, G., Elsken, T., Hutter, F., & Grabocka, J. (2023). Transfer NAS with Meta-learned Bayesian Surrogates. Retrieved May 20, 2023, from https://openreview.net/forum?id=paGvsrl4Ntr

Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., & Summers, R. M. (2016). Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning [Conference Name: IEEE Transactions on Medical Imaging]. *IEEE Transactions on Medical Imaging*, *35*(5), 1285–1298. https://doi.org/10.1109/TMI.2016.2528162

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay [arXiv:1803.09820 [cs, stat]]. https://doi.org/10.48550/arXiv.1803.09820

Smith, L. N. (2022). General Cyclical Training of Neural Networks [arXiv:2202.08835 [cs, stat]]. https://doi.org/10.48550/arXiv.2202.08835

Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2018). Don't Decay the Learning Rate, Increase the Batch Size [arXiv:1711.00489 [cs, stat]]. https://doi.org/10.48550/arXiv.1711.00489

Sui, X., Lv, Q., Zhi, L., Zhu, B., Yang, Y., Zhang, Y., & Tan, Z. (2023). A Hardware-Friendly High-Precision CNN Pruning Method and Its FPGA Implementation [Number: 2 Publisher: Multidisciplinary Digital Publishing Institute]. *Sensors*, *23*(2), 824. https://doi.org/10.3390/s23020824

Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks [arXiv:1905.11946 [cs, stat]]. https://doi.org/10.48550/arXiv.1905.11946

Tanaka, H., Kunin, D., Yamins, D. L. K., & Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow [arXiv:2006.05467 [cond-mat, q-bio, stat]]. https://doi.org/10.48550/arXiv.2006.05467

Taylor, R., Ojha, V., Martino, I., & Nicosia, G. (2021). Sensitivity Analysis for Deep Learning: Ranking Hyper-parameter Influence [ISSN: 2375-0197]. *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 512–516. https://doi.org/10.1109/ICTAI52525.2021.00083

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need [arXiv:1706.03762 [cs]]. https://doi.org/10.48550/arXiv.1706.03762

Wang, B., Shi, W., & Miao, Z. (2015). Confidence Analysis of Standard Deviational Ellipse and Its Extension into Higher Dimensional Euclidean Space. *PLoS ONE*, *10*(3), e0118537. https://doi.org/10.1371/journal.pone.0118537

Wang, C., Zhang, G., & Grosse, R. (2020). Picking Winning Tickets Before Training by Preserving Gradient Flow [arXiv:2002.07376 [cs, stat]]. https://doi.org/10.48550/arXiv.2002.07376

Wang, H., Qin, C., Bai, Y., & Fu, Y. (2023). Why is the State of Neural Network Pruning so Confusing? On the Fairness, Comparison Setup, and Trainability in Network Pruning [arXiv:2301.05219 [cs]]. https://doi.org/10.48550/arXiv.2301.05219

Wang, H., Qin, C., Zhang, Y., & Fu, Y. (2021). Neural Pruning via Growing Regularization [arXiv:2012.09243 [cs]]. https://doi.org/10.48550/arXiv.2012.09243

Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., & Hu, X. (2019). Pruning from Scratch [arXiv:1909.12579 [cs]]. https://doi.org/10.48550/arXiv.1909.12579

Weerts, H. J. P., Mueller, A. C., & Vanschoren, J. (2020). Importance of Tuning Hyperparameters of Machine Learning Algorithms [arXiv:2007.07588 [cs, stat]]. https://doi.org/10.48550/arXiv.2007.07588

Yang, G., & Hu, E. J. (2022). Feature Learning in Infinite-Width Neural Networks [arXiv:2011.14522 [cond-mat]]. https://doi.org/10.48550/arXiv.2011.14522

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., & Gao, J. (2022). Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer [arXiv:2203.03466 [cond-mat]]. https://doi.org/10.48550/arXiv.2203.03466

Yu, T., & Zhu, H. (2020). Hyper-Parameter Optimization: A Review of Algorithms and Applications [arXiv:2003.05689 [cs, stat]]. https://doi.org/10.48550/arXiv.2003.05689

Zhu, M., & Gupta, S. (2017). To prune, or not to prune: Exploring the efficacy of pruning for model compression [arXiv:1710.01878 [cs, stat]]. https://doi.org/10.48550/arXiv.1710.01878