Improving Saliency Metrics for Channel Pruning of Convolutional Neural Networks

by

Kaveena Devi Persand

Dissertation

Submitted to the School of Computer Science and Statistics
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
(Computer Science)

School of Computer Science and Statistics

TRINITY COLLEGE DUBLIN

October 2022

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

. . . .

Kaveena Devi Persand

Dated: October, 2022

General Data Protection Regulation

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

. . . .

Kaveena Devi Persand

Dated: October, 2022

Abstract

Channel pruning is an effective technique to reduce the size of Convolutional Neural Networks (CNNs). A decisive part of any pruning algorithm is its saliency metric. We propose different techniques to improve saliency metrics for channel pruning.

Saliency metrics are encompassed in a larger pruning algorithm and are expressed in various forms. Without a standard form, it can be difficult to identify and compare these metrics. To facilitate the comparison of saliency metrics, we propose a taxonomy based on four independent components: base input, pointwise metric, reduction, and scaling. We classify existing saliency metrics according to our proposed taxonomy. We find that new channel saliency metrics can be created using the components of existing saliency metrics. We also propose a new scaling method. We evaluate the newly created saliency metrics (using existing components as well as our new scaling method) and find that some of the metrics outperform existing ones. We also provide some guidance for the construction of new saliency metrics. Specifically, we highlight the importance of the reduction and scaling methods.

Pruning algorithms generally rely on a single saliency metric for pruning. Even if that chosen metric performs well on average, it can make poor decisions from time to time. Through the use of multiple saliency metrics, we can compensate the poor decisions of the single metric. We show that the combination of saliency metrics is possible and combine the decisions of multiple saliency metrics using a *myopic oracle*. We show that the decisions of the myopic oracle can lead to better pruning rates than the constituent metrics.

When pruning one channel from CNNs with split and join connections, more pruning opportunities become apparent. Multiple channels can be pruned by transitively removing channel weights from other layers of the network. However, most saliency metrics do not factor in these extra structural constraints. We propose *domino* saliency metrics, built on top of existing channel saliency metrics, to factor in these constraints. We show that the use of domino saliency metrics can significantly improve pruning rates for networks with splits and joins.

Acknowledgements

This PhD would not have been possible without the support of those who directly or indirectly contributed to it.

Firstly, I would like to thank Science Foundation Ireland and Arm Research for financially supporting this PhD through the grant 13/RC/2094.

I also thank my family and friends (including the cats of Finglas) for supporting me throughout my journey into research. I greatly appreciated your help in proof-reading my thesis, Bhuvan.

I would like to thank Andrew, my colleague from Córais/Systems Tools Group and co-author on my first-author papers, for helping with your knowledge on systems and machine learning. I have learned a lot from you, including computer science, scientific writing, software tools, or even great food spots in Dublin.

I would like to thank all my other colleagues from the Córais/Systems Tools Group: Asad, Basia, Cormac, James, and Yuan. Through our numerous discussions, I have learned a lot about research and computer science.

Finally, I would like to thank my supervisor, David, without whom this PhD would not have been possible. You have been a great supervisor, especially during a pandemic that changed all of our lives. You always gave great advice whether it was in regards to finding new research topics, going through the peer review process, writing a thesis, my teaching endeavours, or just navigating life as a PhD student. I would like to thank you again for your guidance and multiple corrections for this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Motivation

Convolutional Neural Networks (CNNs) that achieve high accuracy values on image classification tasks are often too large to be deployed on highly memory-constrained devices such as mobile phones. Channel pruning is an effective technique to reduce the size of these large networks. Pruning aims to remove weights whose removal are least likely to damage the network. The pruning algorithm uses a saliency metric to identify the weights to be pruned. The saliency metric is, thus, an essential component of pruning algorithms. Hence, improving saliency metrics for channel pruning can help push the boundaries of pruning further.

1.1 Challenges in Improving Saliency Metrics for Channel Pruning

We identified that though the literature on saliency metrics for channel pruning is extensive, there are still many open questions. We aim to investigate a number of these avenues in the hopes of improving saliency metrics for pruning.

- **Can we express saliency metrics in a standard form?**

A large number of saliency metrics have been proposed. These saliency metrics often share commonalities with other existing saliency metrics but finding these commonalities can be difficult. These difficulties arise due to the use of different forms to express the metric and the metric, itself, being encompassed in a larger pruning algorithm. One of the first questions that we want to tackle is whether we can find a standard form to express the existing saliency metrics for channel pruning.

- **Given a standard form for saliency metrics, how do we construct good metrics?**

If we can identify a standard form to saliency metrics, we want to further investigate the different aspects of saliency metrics. Do certain aspects of saliency metrics lead to finding smaller networks? If so, what are they and can these choices be used to find better metrics than the existing ones?

- **Can multiple saliency metrics be used to find better decisions than a single metric?**

Saliency metrics are commonly built on some assumption about the network. These assumptions may not be valid throughout the pruning process. Hence, even good saliency metrics may make poor decisions from time to time. However, most pruning algorithms rely on a single saliency metric for the entire pruning process. By relying on a single albeit good saliency metric, we risk occasionally missing better pruning options. Hence, an obvious question is whether the use of multiple saliency metrics can be used to compensate the poor pruning decisions of these otherwise good metrics and improve pruning.

- **How can channel saliency metrics incorporate information about other channels that may be removed?**

Many high accuracy networks for image classifications contain split and join connections. The presence of these split and join connections create structural dependencies that are often exploited during pruning. However, most saliency metrics do not explicitly incorporate these structural information into them. For networks with branches, the lack of structural information in the saliency metric used for pruning can be detrimental. We, hence, investigate how to extend existing saliency metrics to include information about these structural dependencies.

1.2 Contributions

In Chapter 2, we give an overview of CNNs and explain the notation that we use for CNN description and pruning. We break down pruning algorithms according to seven different aspects: granularity of pruning, domain of weights, distribution of pruning rates, repair strategy, pruning schedule, hard or soft pruning, and saliency metric. We review state-of-the-art pruning algorithms according to these seven aspects. We also give a literature review on the use of penalty terms, masks, and AutoML for pruning.

In Chapter 3, we tackle the challenge of comparing saliency metrics. We propose a taxonomy for classifying saliency metrics using four independent components: base input, pointwise metric, reduction, and scaling. We classify existing channel saliency metrics using our taxonomy. We also propose a new scaling method. By combining components of previously established saliency metrics and our new scaling method, we find channel saliency metrics that outperform published metrics. A modified version of Chapter 3 has been published in IEEE Access.

Kaveena Persand, Andrew Anderson [1] and David Gregg, "Taxonomy of Saliency Metrics for Channel Pruning," in IEEE Access, vol. 9, pp. 120110-120126, 2021, doi: 10.1109/ACCESS.2021.3108545.

In Chapter 4, we aim to combine multiple saliency metrics. We propose a *myopic oracle* to combine the decisions of multiple saliency metrics. We find that the combined decision of the myopic oracle can be used to improve channel pruning. A modified version of Chapter 4 has been published in 2020 IEEE Symposium Series on Computational Intelligence.

Kaveena Persand, Andrew Anderson [1] and David Gregg, "Composition of Saliency Metrics for Pruning with a Myopic Oracle," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), 2020, pp. 753-759, doi: 10.1109/SSCI47803.2020.9308157.

In Chapter 5, we investigate how to incorporate structural information into channel saliency metrics for pruning. We propose *domino* saliency metrics to include

[1] I, Kaveena Persand, am the main author of the software, scientific discussion and writing for the work presented in this thesis. Dr Andrew Anderson contributed to the scientific discussion and writing of these papers. He also helped configure the software tools used for our experimental evaluation.

structural information into channel saliency metrics. Our domino saliency metrics are built using existing channel saliency metrics. Using domino saliency metrics, we combine the saliency of channels that are found to participate in split and join connections. We show that our domino saliency metrics can be used to improve channel pruning for networks with branches. A modified version of Chapter 5 has been accepted for publication at 20th International Conference of the Italian Association for Artificial Intelligence (AIxIA 2021) in Springer LNAI.

Finally, we discuss future work in Chapter 6 and give our final thoughts in Chapter 7.

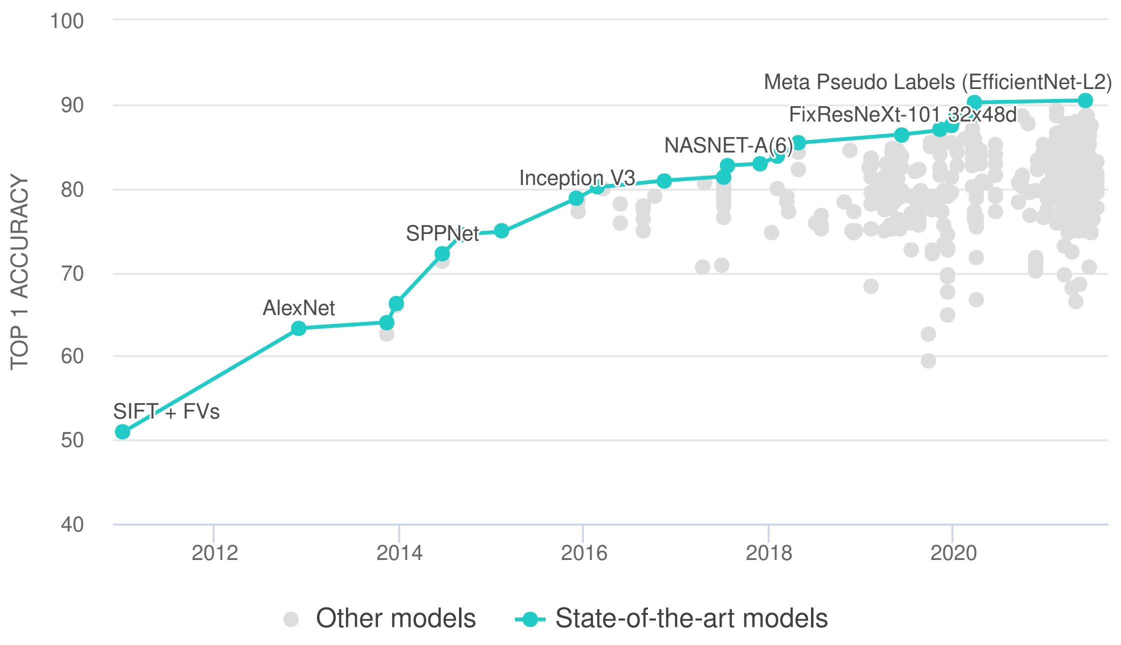
Chapter 2

Background & Literature Review of Pruning Convolutional Neural Networks

2.1 Introduction

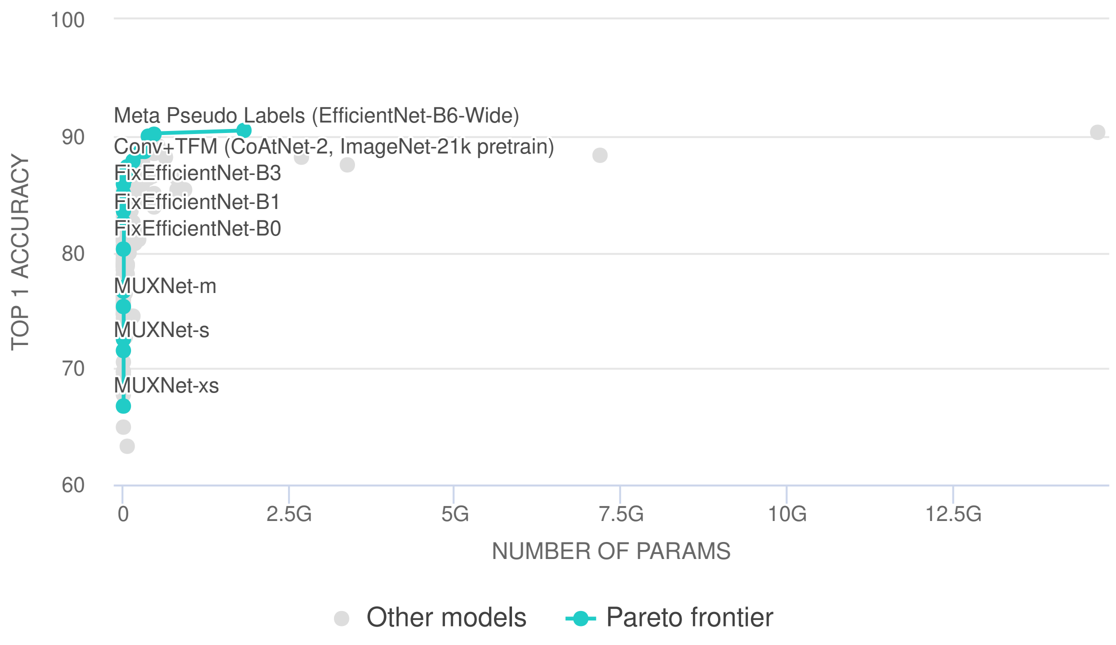
State-of-the-art Deep Neural Networks (DNNs) can achieve high accuracy on complex image classification tasks (Hestness et al. 2019; He et al. 2015). Figure 2-1a shows the state-of-the-art DNNs for the recent ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Deng et al. 2009) challenges. As shown by the Pareto optimal points in Figure 2-1b, recent architectures with over 90% classification accuracy on the ILSVRC classification task contain over a billion parameters (Zhai et al. 2021; Pham et al. 2021; Riquelme et al. 2021). Figure 2-1b shows the accuracy for DNNs with different numbers of parameters with the most accurate model using almost 2.5 billion parameters. While deploying a large model is not necessarily a concern on data centres, embedded devices have significant memory constraints. According to GSMArena, the average amount of RAM for mobile phones was slightly over 4 gigabytes in 2018 (*Counterclockwise: RAM capacity through the years* n.d.). Hence, deploying models with over 2.5 billion parameters of 4 bytes each would be impossible on the average mobile phone.

State-of-the-art models need to be adapted for deployment on embedded devices. The two most common techniques for DNN model compression are pruning



(a) Evolution of DNN accuracy for ILSVRC ImageNet classification challenge.

Source: `https://paperswithcode.com/sota/image-classification-on-imagenet` (Last Accessed on 05-August-2021)



(b) Top-1 accuracy for DNNs with different number of parameters for ILSVRC ImageNet classification challenge.

Source: `https://paperswithcode.com/sota/image-classification-on-imagenet?dimension=Number%20of%20params` (Last Accessed on 05-August-2021)

Figure 2-1: Evolution of classification accuracy of DNNs for the ILSVRC challenge.

and quantisation (Liang et al. 2021a; Menghani 2021). Pruning is the removal of parameters (also called weights) of the network from the DNN. Pruning weights reduces the number of parameters to be stored for the network. Pruning is a very

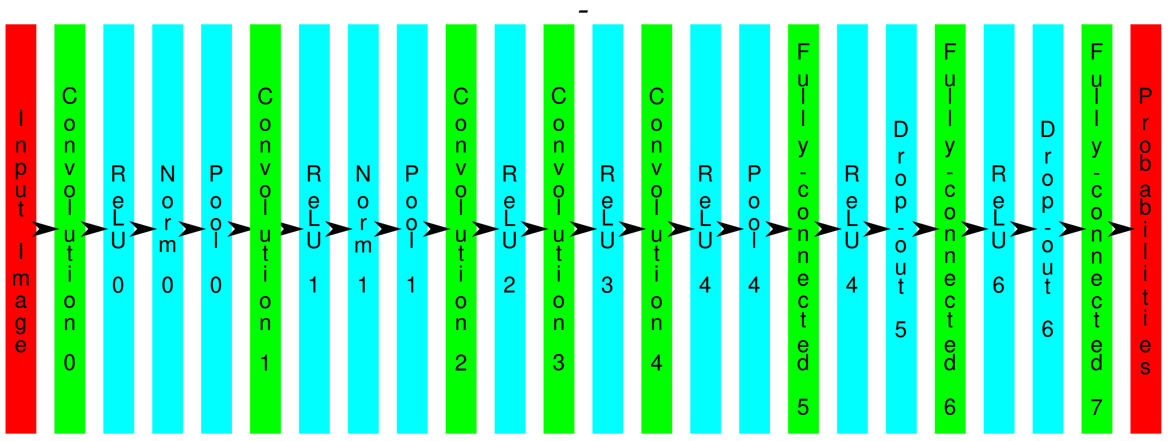


Figure 2-2: Layer-wise structure of AlexNet.

versatile technique and can be used along with other network compression techniques to obtain smaller networks (Han et al. 2016; Chen et al. 2018; Noy et al. 2020).

2.2 Convolutional Neural Networks

The most common type of DNN for image classification are CNNs (Convolutional Neural Networks). AlexNet (Krizhevsky et al. 2017), LeNet (Lecun et al. 1998), GoogleNet (Szegedy et al. 2015), VGG (Simonyan and Zisserman 2014), and ResNets (He et al. 2016) are examples of popular convolutional neural networks. Convolutional neural networks are neural networks with convolution layers.

To explain what a convolution layer is, we first need to introduce the layer-wise structure of neural networks.

Neural networks are represented by composing together many different functions (Goodfellow et al. 2016). If $\mathcal{Y}(x)$ is the classification function of a simple forwardfeed neural network, x its input, and $\mathcal{Y}(x) = y^3(y^2(y^1(x)))$, then y^1, y^2, and y^3 are called layers of the network with y^1, the input layer and y^3 the final layer. Deeper layers are further away from the input of the network.

The layer-wise structure of a CNN, AlexNet (Krizhevsky et al. 2017), is shown in Figure 2-2. The input of the CNN is "Input Image" and its output is "Probabilities". The layers coloured in green (convolution and fully-connected layers) are layers containing permanent parameters called weights. The layers coloured in blue are activation layers. Activation layers are typically non-linear functions that are applied between convolution or fully-connected layers.

2.2.1 Notation

We introduce some notation to facilitate the description of a CNN.

Weights

Let W represent the vector of all trained parameters of a given CNN. $\widetilde{W}$ represents the pruned parameters vector. Let $n(\cdot)$ represent the cardinality of a tensor or vector, then $n(W) > n(\widetilde{W})$.

Layer-wise notation

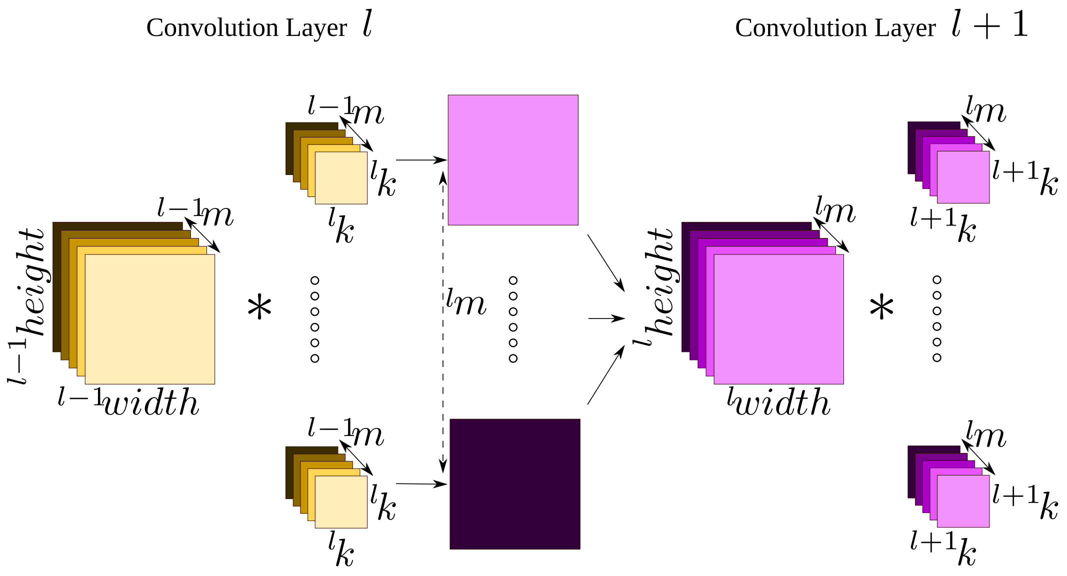


Figure 2-3: Weights and feature map structure in a convolutional neural network.

The output of a layer, l, is denoted ^{l}A. The weights of a layer are denoted ^{l}W. Throughout this thesis we use a superscript on the left to indicate the layer targeted.

The parameters and feature maps of CNNs are represented using multi-dimensional tensors. Feature maps are 3-dimensional tensors comprised of multiple 2-dimensional feature maps. Each 2-dimensional output feature map of the l^{th} layer, has $^{l}height \times {}^{l}width$ individual points called pixels. The number of 2-dimensional feature maps produced by the layer is ^{l}m and the number of 2-dimensional input feature maps used by the layer is ^{l}c. Since convolution layers are often chained, ^{l}c is often equal to the number of output feature maps of the previous layer ^{l-1}m.

Each 2-dimensional input feature map is convolved with its corresponding 2-dimensional kernel of shape ${}^{l}k \times {}^{l}k$ and summed together to create a single 2-dimensional output feature map. Convolution weight tensors are 4-dimensional tensors comprised of $({}^{l}m \times {}^{l}c)$ 2-dimensional kernels.

The structures of the input feature maps, output feature maps, and weight tensor are illustrated in Figure 2-3.

Channel-wise notation

The c^{th} feature map of ${}^{l}A$ is denoted ${}^{l}A_c$. The weights of the c^{th} output channel are denoted ${}^{l}W_c$. In Figure 2-4, a feature map ${}^{l}A_c$ is highlighted in blue and the weights of the same channel are highlighted in red.

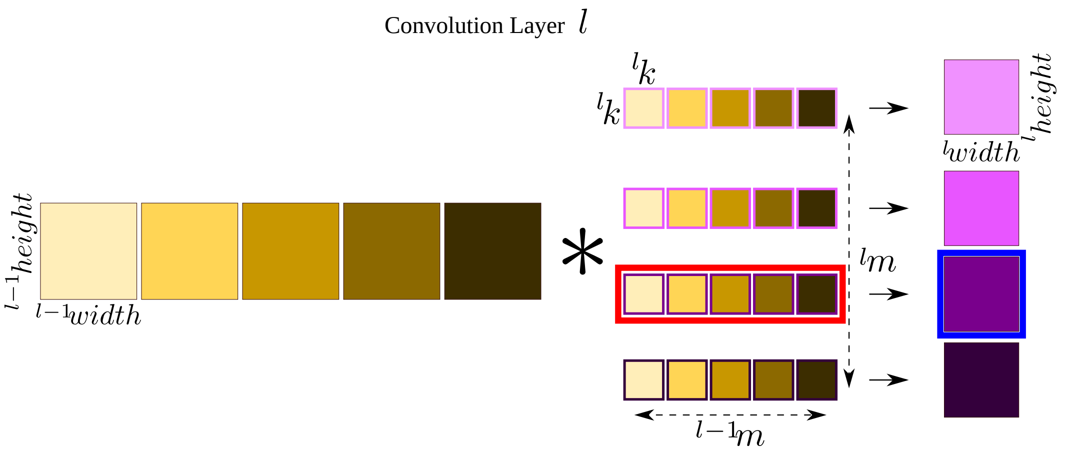


Figure 2-4: Feature and weights associated with a single convolution channel.

Slices of Tensors

Feature maps and weights of convolution layers are both presented as multidimensional tensors. The layer-wise and channel-wise notations allow us to easily represent slices of these tensors.

To represent an individual element of the weight tensor, we use 4 independent indices. ${}^{l}W[m, c, u, v]$ is a single weight from the 4-dimensional weight tensor with $0 \leq m < {}^{l}m - 1$, $0 \leq c < {}^{l}c - 1$, $0 \leq u < {}^{l}k - 1$, and $0 \leq v < {}^{l}k - 1$. Similarly, ${}^{l}A[m, h, w]$ represents a single pixel from a 3-dimensional feature map with $0 \leq m < {}^{l}m - 1$, $0 \leq h < {}^{l}height - 1$, and $0 \leq w < {}^{l}width - 1$.

We also use a concise notation to describe slices of a tensor using the ":" notation. The ":" notation is used to indicate that all the valid indices are being considered. For example, $W[m, c, k, :] \equiv W[m, c, k, 0..{}^{l}k - 1]$. Hence, with our channel notation, ${}^{l}W_c \equiv {}^{l}W[:, c, :, :]$ and ${}^{l}A_c \equiv {}^{l}A_c[c, :, :]$.

${}^{l}W_c$ is a slice of shape ${}^{l}m \times {}^{l}k \times k$ from ${}^{l}W$ and ${}^{l}A_c$ is a slice of shape ${}^{l}height \times {}^{l}width$ from ${}^{l}A$.

Loss

CNNs are trained by optimising the weights of the network according to a loss function. The most common loss function for modern CNNs is the cross-entropy loss. The equation of the cross-entropy loss, $\mathcal{L}$, is given by Equation 2.1 for a network with weights W and input dataset $\mathcal{I}_{set}$. The dataset $\mathcal{I}_{set}$ contains N pairs of input images, ${}_{n}I$, and labels (or true vector of probabilities classifying ${}_{n}I$), ${}_{n}L$. P gives the output of the network, i.e., the vector of probabilities classifying the input image. $P(W, {}_{n}I)$ is a vector containing one probability for each possible class.

$$\mathcal{L}(W, \mathcal{I}_{set}) = \sum_{n=0}^{N-1} {}_{n}\mathcal{L}(W) = \sum_{n=0}^{N-1} -({}_{n}L \odot log(P(W, {}_{n}I))) \tag{2.1}$$

2.2.2 Convolution Layer

A simple multi-channel multi-kernel 2-dimensional (Vasudevan et al. 2017) convolution layer, l, is given in Equation 2.2 where ${}^{l-1}A$ is the output of the previous layer. In image classification, convolution layers canonically have multiple 2-dimensional input images and multiple 2-dimensional output images. In practice, the only layer with a single input image is the input layer in the case of monochrome image classification. If a 2-dimensional image is an intermediate result in the network, it is referred to as a feature map instead of an image. A convolution layer can be broken down into multiple 2-dimensional convolutions. Each output feature map is the addition of the result of the convolution of each input feature map with its corresponding 2-dimensional filter. These 2-dimensional filters are more commonly referred to as kernels in the context of CNNs. If a layer has ${}^{l}c$ input feature maps and ${}^{l}m$ output feature maps then $({}^{l}m \times {}^{l}c)$ 2-dimensional convolutions are conducted for that convolution layer. In Equation 2.2, ${}^{l}A[m, :, :]$ represents one 2-dimensional output feature map and ${}^{l-1}A[c,:,:]$ one 2-dimensional input feature map. ${}^{l}W$ are the parameters, i.e. weights, of the layer.

$$ {}^{l}A[m,h,w] = \sum_{c=0}^{{}^{l}c-1} \sum_{u=0}^{{}^{l}k-1} \sum_{v=0}^{{}^{l}k-1} {}^{l-1}A[c,h+u,w+v]\,{}^{l}W[m,c,u,v] \tag{2.2} $$

Figure 2-5 illustrates the multiplications and additions performed to compute a single output pixel of a single output feature using Equation 2.2. To compute one output pixel, the 2-dimensional kernels are multiplied with their corresponding slices of the input feature maps and added together to create intermediate output points. These intermediate output points are then added across the different channels to obtain the final output point.

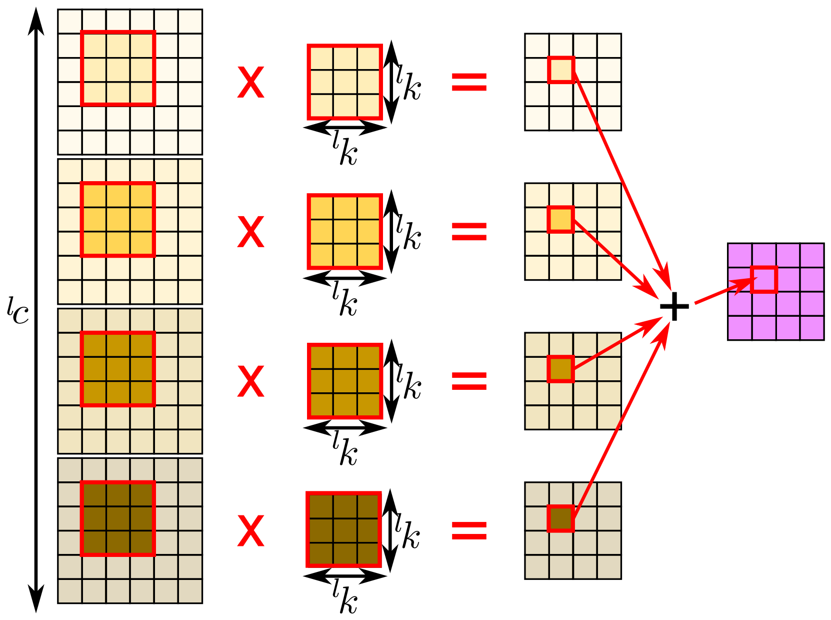


Figure 2-5: Weights and feature maps used to compute a single output point of a convolution layer.

2.2.3 Fully-connected layer

A simple fully-connected layer is given in Equation 2.3 where ${}^{l-1}A$ is the output of the previous layer. Fully-connected layers have 1-dimensional vectors as input and 1-dimensional vectors as output.

$$^{l}A[m] = \sum_{c=0}^{^{l}c-1} {}^{l-1}A[c]W^{l}[m, c] \tag{2.3}$$

2.3 Pruning

Pruning is the removal of weights from the network. Since its use by LeCun et al. (1989), Mozer and Smolensky (1988), Hassibi and Stork (1992), and Karnin (1990) for ANNs, pruning has been adapted for use in modern CNNs (Han et al. 2015).

The degree of pruning of a network is measured by its pruning rate. The pruning rate is also called sparsity level. The pruning rate is the percentage of weights removed compared to the unpruned network. The definition of pruning rate, $\mathcal{P}$, using $\widetilde{W}$ and W is given in Equation 2.4.

$$\mathcal{P} = 1 - \frac{n(\widetilde{W})}{n(W)} \tag{2.4}$$

The removal of weights from the network often results in lower memory requirements (Han et al. 2015; Mao et al. 2017; Anwar et al. 2017; Molchanov et al. 2017). Han et al. (2016) show that pruning can significantly reduce the memory requirement of a network while maintaining its classification accuracy.

Pruning can also be used to reduce the number of operations to be performed when running the network. He et al. (2018a) and Luo et al. (2017) show that pruning results in fewer Floating Point Operations per Second (FLOPS).

Lin et al. (2018), Turner et al. (2018), and Yao et al. (2018) show that pruning can also be used to accelerate the network.

Pruning can also improve generalisation of networks (Mozer and Smolensky 1988; LeCun et al. 1989; Hassibi and Stork 1992; Han et al. 2015; Wen et al. 2016; You et al. 2019; Frankle and Carbin 2019; Liu et al. 2017b).

2.4 Classification of Pruning Algorithms

Numerous pruning algorithms have been proposed, with similarities and differences from existing pruning algorithms (Blalock et al. 2020). In this section, we explore different ways of classifying these algorithms.

Reed (1993) categorised early pruning algorithms into two categories depending on how they identify unimportant weights: using sensitivity and using penalty terms. Sensitivity is the change in the loss function caused by pruning (Mozer and Smolensky 1988). We discuss sensitivity and its approximations in more details in Section 3.7.2. Penalty terms are terms introduced in the loss function to drive unimportant weights to zero. We further discuss pruning algorithms using penalty terms in Section 2.5.1. This early classification fails to encompass new pruning algorithms which may vary in more aspects than how to identify unimportant weights. A summary of these newer classifications and how they relate to each other is given in Figure 2-6.

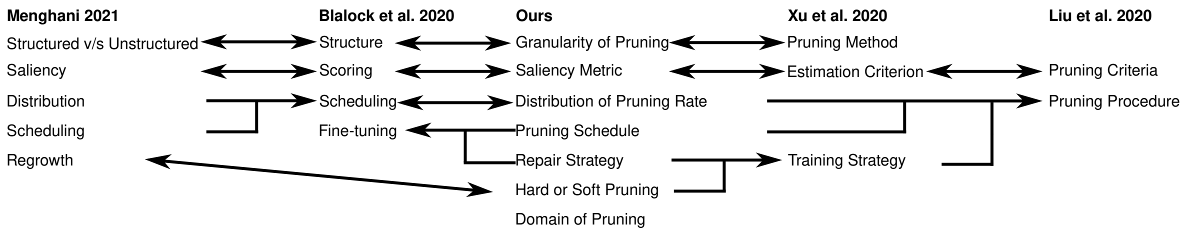


Figure 2-6: Different aspects in which pruning algorithms can vary according to Xu et al. (2020), Menghani (2021), Liu et al. (2020a), and Blalock et al. (2020) and how they relate to our classification.

Pruning algorithms can be broken down into several components. Liu et al. (2020a), Xu et al. (2020), Blalock et al. (2020), and Menghani (2021) identify that pruning algorithms use an estimation criteria, scoring or saliency metric to identify unimportant parameters.

Along with the estimation criterion Xu et al. (2020) identify a pruning method and training strategy. Xu et al. (2020) classify pruning methods as being either non-structured or structured. The training strategy defines how training and retraining is used with pruning. Xu et al. (2020) classify the training strategy in three categories: hard pruning, soft pruning, and redundant learning strategy.

Menghani (2021) identifies five aspects to the pruning algorithm: saliency metric, structured or unstructured, distribution, scheduling, and regrowth. The saliency metric identified by Menghani (2021) is similar to the estimation criterion identified by Liu et al. (2020a) and Xu et al. (2020). The distribution describes how pruning rates should be distributed in the network. Scheduling describes how pruning is mixed with the training strategy. Regrowth defines whether pruned weights are permanently removed or not.

Blalock et al. (2020) identify four aspects in which pruning algorithms differ: structure, scoring, scheduling, and fine-tuning.

Following the different aspects of the pruning algorithm explored by Reed (1993), Xu et al. (2020), Liu et al. (2020a), and Menghani (2021), we group pruning algorithms for convolutional neural networks according to the following aspects:

- Granularity of pruning
- Domain of pruning
- Distribution of pruning rates
- Pruning schedule
- Repair Strategy
- Hard or soft pruning
- Saliency metric

In Figure 2-6, we summarise how the different aspects of the pruning algorithms identified by Xu et al. (2020), Menghani (2021), Liu et al. (2020a), and Blalock et al. (2020) relate to each other and to our own classification.

2.4.1 Granularity of Pruning

Pruning was a technique initially described for fully-connected layers (LeCun et al. 1989; Mozer and Smolensky 1988; Hassibi and Stork 1992) and has been extended to convolution layers (Han et al. 2015).

Convolution layers and fully-connected layers are highly structured layers with each input pixel contributing to more than one output pixel and each output pixel using the contribution of more than one input pixel. Weights of these layers are structured to reflect this relationship. For a convolution layer with ${}^{l}m$ output feature maps and ${}^{l}c$ input feature maps using 2-dimensional kernels of shape ${}^{l}k \times {}^{l}k$, the weight tensor is of the shape ${}^{l}m \times {}^{l}c \times {}^{l}k \times {}^{l}k$.

The removal of weights from the network can be done in a structured or unstructured way. The pruning algorithm removes multiple pruning elements from the network. A pruning element is a pattern of weights defined by the granularity of pruning. The patterns of weights removed can be categorised into different granularities of pruning (Mao et al. 2017).

For fully-connected layers, the common granularities of pruning are either fine-grain (LeCun et al. 1989) or neuron-wise (Mozer and Smolensky 1988). Figure 2-7 shows the ${}^{l}m \times {}^{l}c$ structure of a fully-connected layer's weight tensor and pruning granularities.

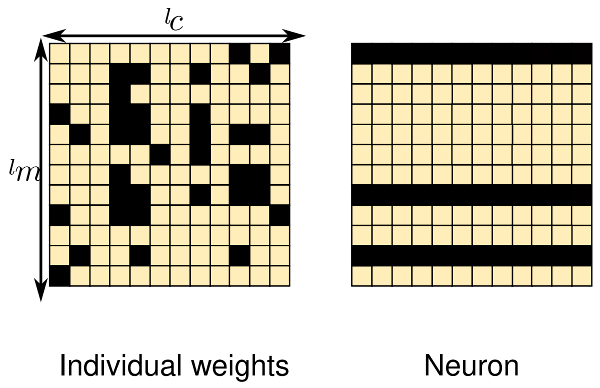


Figure 2-7: Different granularities of pruning for a fully-connected layer. The dark squares represent pruned parameters.

According to Wen et al. (2016), Mao et al. (2017), and Anwar et al. (2017), the different granularities of pruning for convolution layers can be categorised into: fine-grain, intra-kernel, kernel-wise, and channel-wise. Figure 2-8 shows the structure of these four granularities of pruning. Weights of convolution layers are structured in a tensor with shape ${}^{l}m \times {}^{l}c \times {}^{l}k \times {}^{l}k$. The weight tensor can be pruned according to different granularities: fine-grain (also called individual weights), intra-kernel, kernel, and coarse-grain (also called channel). These four structures of pruning are illustrated with black squares representing pruned weights.

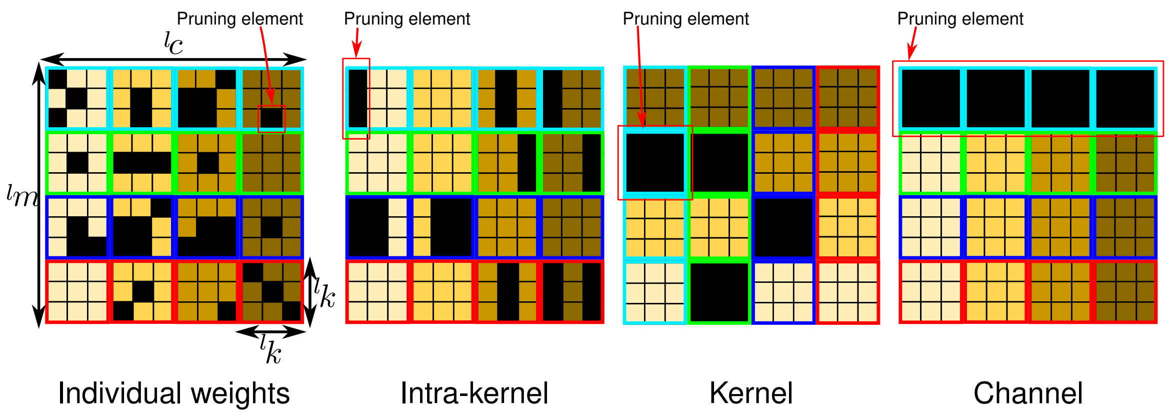


Figure 2-8: Different granularities of pruning for a convolution layer. The dark squares represent pruned parameters.

Fine-grain Pruning

Fine-grain pruning removes individual parameters without any constraint in the pattern of the weights removed. Fine-grain pruning is also called unstructured pruning. The early use of fine-grain pruning was described for fully-connected layers by LeCun et al. (1989) using a second order Taylor expansion on a fully-trained ANN (Artificial Neural Network).

Fine-grain pruning was popularised for CNNs by Han et al. (2015) and has since been extensively used for CNN pruning (Mao et al. 2017; Han et al. 2017; He et al. 2018b; Frankle and Carbin 2019; Guo et al. 2016; Wang et al. 2020b; Ding et al. 2019; Liu et al. 2018a; Li et al. 2017b).

Fine-grain pruning can achieve very high pruning rates (Han et al. 2015; Mao et al. 2017; Han et al. 2017; He et al. 2018b; Frankle and Carbin 2019; Guo et al. 2016). However, the weight tensors need to be stored in a compressed format such as Compressed Sparse Row (CSR) for these pruning rates to result in memory savings (Mao et al. 2017).

Coarse-grain Pruning

Coarse grain pruning was first introduced for fully-connected layers as the removal of entire units, i.e. neurons, from the network by Mozer and Smolensky (1988). In fully-connected layers, this results in the removal of an entire row or column of the weight matrix (Mao et al. 2017). The removal of neurons in the final layer, results in the removal of an output class. Hence, pruning of neurons in the final layer is generally avoided.

In CNNs, coarse-grain pruning is the removal of entire convolution channels or filters from the layer (Wen et al. 2016; Mao et al. 2017; Molchanov et al. 2017; Anwar et al. 2017). It is also called feature map pruning (Anwar et al. 2017).

The rightmost illustrations in Figures 2-7 and 2-8 respectively show the pattern of weights removed from a fully-connected layer and a convolution layer when coarse-grain pruning is applied.

Pruning entire channels from convolution layers also has the advantage of producing a smaller dense weight tensor (Yu et al. 2017), which allows decades of research on dense linear algebra performance to continue to be applied. The smaller weight tensor can be used to accelerate convolutional neural networks (Wen et al. 2016; Lebedev and Lempitsky 2016; You et al. 2019; He et al. 2017; Polyak and Wolf 2015; Turner et al. 2018).

Other Granularities of Pruning

Fine-grain pruning and coarse-grain pruning are the two extremes of pruning strategies. Mao et al. (2017) and Anwar et al. (2017) explore the intermediate granularities between fine-grain and coarse-grain pruning in convolution layers. Lebedev and Lempitsky (2016) propose regular interval pruning of the weight tensor for convolution layers that are implemented using im2col (Jia et al. 2014; Chetlur et al. 2014; Donahue et al. 2014).

2.4.2 Domain of Weights

The convolution operation presented in Equation 2.2 assumes that the operands are represented in the spatial domain, i.e. the normal image space. Fast convolution algorithms that convert the convolution operation and its weights into another domain can be used to accelerate the network (Lavin and Gray 2016). Lavin and Gray (2016) show that convolutions can be accelerated using Fast Fourier Transforms and Winograd transforms. For these transformed convolutions, the weights also need to be transformed. The transformed weights can also be used for pruning. Liu et al. (2018a) and Li et al. (2017b) prune the weights in the Winograd domain. Liu et al.

(2018b) prune weights in the frequency domain. Yu et al. (2019) prune the weights in the spatial domain (before transformation) and in the Winograd domain (after transformation) to boost pruning rates.

2.4.3 Distribution of Pruning Rates

The "distribution" of pruning rates describes how many pruning elements (see Section 2-8) should be removed at each pruning iteration. It describes how the pruning rates should be allocated throughout the network or the pruning process. While it is common for pruning algorithms to explicitly set the number of elements to be removed at each pruning iteration (LeCun et al. 1989; Molchanov et al. 2017; Theis et al. 2018; Molchanov et al. 2019; Srinivas and Babu 2015), it can also be derived to meet another constraint (Han et al. 2015; Yu et al. 2018; Guo et al. 2016; Hu et al. 2016). Distribution of pruning rates can be classified into two distinct groups according to how they allocate pruning rates through the network: *global* and *layer-wise*.

Global Distribution of Pruning Rates

Algorithms that use a global distribution of pruning rates prune weights irrespective of the layer in which they appear.

Lin et al. (2018) set a global sparsity threshold and use a global saliency metric to prune channels. LeCun et al. (1989), Molchanov et al. (2017), and Theis et al. (2018) each remove a single pruning element from the entire network at every pruning iteration. LeCun et al. (1989) remove one individual weight at each pruning iteration whereas Molchanov et al. (2017) and Theis et al. (2018) remove one entire convolution channel at each pruning iteration. Srinivas and Babu (2015) perform a sensitivity analysis prior to the pruning process to determine the number of elements to remove from the network at each pruning iteration. Molchanov et al. (2019) remove a predefined number of neurons at each pruning iteration.

Instead of directly defining the number of pruning elements to be removed, global pruning algorithms can use other constraints set by the user to derive the pruning rate. Han et al. (2015) prune all weights that have a magnitude below a global threshold. Yu et al. (2018) set a maximum degradation for the final layer of the networks and prune the maximum number of channels such that the estimated damage to the network is bound by an upper limit.

Layer-wise Distribution of Pruning Rates

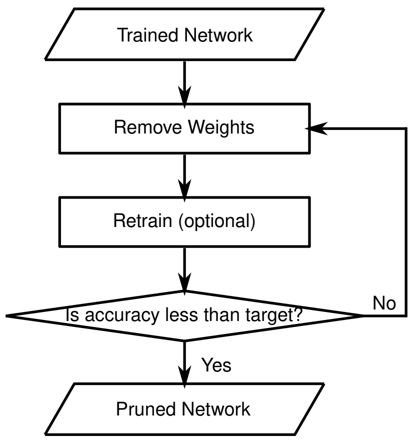
Algorithms with a layer-wise distribution of pruning rates consider the pruning elements of different layers to be part of different pools for pruning. The pruning rates are hence allocated per layer and not per network. A global pruning rate of 10% does not give any information about the layer-wise pruning rates. The weights can be removed from the same layer or from different layers. A pruning rate of 10% per layer ensures that all layers are pruned while removing the same number of weights from the network as a global pruning rate of 10%.

Polyak and Wolf (2015) and Mao et al. (2017) perform a sensitivity analysis prior to the pruning process to determine pruning rates for different layers. He et al. (2017) use different user-defined percentage of weights to be removed depending on the position of the layer with deeper layers being pruned less aggressively.

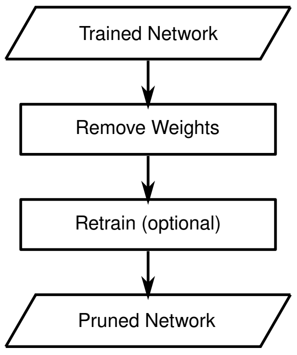
Layer-wise pruning algorithms can also use other user-defined constraints to derive the layer-wise pruning rates. He et al. (2018b) use user-defined global pruning rates and use reinforcement learning to determine the layer-wise pruning rates. Guo et al. (2016) prune all weights that have an absolute value below a layer-wise threshold. Hu et al. (2016) use a layer-wise threshold deduced from the mean average percentage of zeros from the layer to prune channels.

2.4.4 Repair Strategy

The removal of weights from a network decreases its test accuracy (Mittal et al. 2018). The removal of weights is often accompanied by a strategy to repair the network. The most common repair strategy is the use of retraining or fine-tuning (Han et al. 2015; Liu et al. 2017b; Jiang et al. 2018; Molchanov et al. 2017; Molchanov et al. 2019; Theis et al. 2018; Han et al. 2017; Mao et al. 2017; Lin et al. 2018). With retraining, the pruned network is trained for more iterations after the removal of weights. Retraining is also referred to as fine-tuning. Le and Hua (2021) explore the different options of retraining a pruned network.



(a) A simple iterative pruning schedule with a trained network.



(b) A simple single-shot pruning schedule with a trained network.

Figure 2-9: Two common pruning schedules using iterative pruning or single-shot pruning.

Alternative network repair strategies have also been proposed. Hassibi and Stork (1992) and Hassibi et al. (1993), use the Hessian matrix to repair the network. Dong et al. (2017a) use the layer-wise Hessian and error to repair the damaged linear operations. Liang et al. (2021b) repairs the weights by minimising the reconstruction error of the following layer's feature maps. He et al. (2017) find redundant channels using a LASSO regression and reconstruct remaining channels using linear mean squares. Srinivas and Babu (2015) combine redundant neurons together to compensate for their removal.

2.4.5 Pruning Schedule

The pruning schedule defines how pruning interacts with the training process and repair strategy. The pruning schedule determines whether pruning is conducted after, during, or before training the network to convergence. Pruning schedules can be categorised as iterative (using multiple pruning iterations) or single-shot (using a single pruning iteration). During a pruning iteration weights are removed according to the distribution of pruning rates. The removal of weights is very often followed by a repair of the network according to the repair strategy.

A common pruning schedule (LeCun et al. 1989; Han et al. 2015; Hassibi and Stork 1992; Molchanov et al. 2017; Molchanov et al. 2019; Theis et al. 2018) that uses an iterative method and a trained network is shown in Figure 2-9a.

Iterative or Single-shot

Iterative pruning algorithms use multiple pruning iterations until the user-specified stop condition is met.

Iterative pruning algorithms are the most common pruning schedule (Han et al. 2015; Mao et al. 2017; Srinivas and Babu 2015; LeCun et al. 1989; Hassibi and Stork 1992; Molchanov et al. 2017; Molchanov et al. 2019; Theis et al. 2018) at every stage of the pruning process.

In contrast to iterative pruning algorithms, single-shot pruning algorithms use a single pruning iteration to find the final pruned network. A simple single-shot pruning algorithm is shown in Figure 2-9b. Single-shot pruning has also been shown to be effective after (He et al. 2019; Li et al. 2017a) or before (Lee et al. 2019; Wang et al. 2020a) the training process.

Interaction with the Training Process: After or Before

Traditional pruning schemes prune weights after the network has been trained (LeCun et al. 1989; Hassibi and Stork 1992; Mozer and Smolensky 1988; Karnin 1990; Molchanov et al. 2017; Han et al. 2015; Anwar et al. 2017).

Recent approaches show that pruning can be conducted at initialisation with training applied after the pruning process (Frankle and Carbin 2019). Frankle and Carbin (2019) and Frankle et al. (2020) iteratively prune weights at initialisation to find networks with high sparsity. Their algorithm is illustrated in Figure 2-10. Lee et al. (2019) and Lee et al. (2020) use a signal propagation-based approach to prune channels at initialisation using a single shot algorithm. Wang et al. (2020a) use a saliency metric based on gradient conservation and a single shot algorithm to prune the network before training to convergence. Tanaka et al. (2020) show that a data-agnostic method can be used to successfully prune at initialisation.

2.4.6 Hard or Soft Pruning

Hard pruning is the permanent removal of weights from the network. Most pruning algorithms do not allow pruned weights to participate in the network's classification

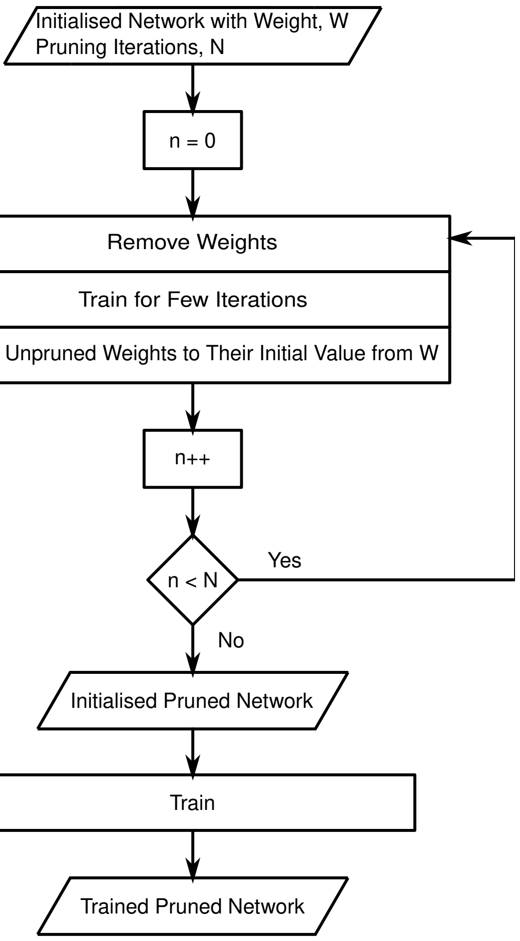

Initialised Network with Weight, W
Pruning Iterations, N
n = 0
Remove Weights
Train for Few Iterations
Reset Unpruned Weights to Their Initial Value from W
n++
n < N
Yes
No
Initialised Pruned Network
Train
Trained Pruned Network

Figure 2-10: The pruning at initialisation algorithm proposed by (Frankle and Carbin 2019).

task after removal (Theis et al. 2018; Li et al. 2017a; He et al. 2017; Luo et al. 2017; Luo et al. 2019).

In contrast to hard pruning, soft pruning allows pruned weights to be brought back at a later stage of the pruning process. He et al. (2018a) allow filters to be unpruned if the weights become salient again. Yu et al. (2017) and Liu et al. (2017b) allow pruned connections to be "unpruned" if the weights become important to the network during the pruning or repair process.

2.4.7 Saliency Metric

As shown in Figure 2-6, saliency metrics are used by a majority of pruning algorithms. The saliency metric (also called estimation criterion or scoring method) is a heuristic used to find the weights whose removal will cause the least damage to the classification accuracy of the network. The saliency metric is used, analogous to a distance measure, to measure and compare the importance of parameters for pruning. To avoid confusion with a similar DNN problem of creating saliency maps, it should be noted that while image saliency (Tomsett et al. 2020; Huang and Gao 2020) and saliency metrics for pruning may share some commonalities in estimating important pixels or parameters, they differ in their aim. Image saliency aims to explain the outputs of image classifiers by creating a saliency map for data scientists to understand the decisions of a network (Tomsett et al. 2020) and is often a subjective task (Huang and Gao 2020). On the other hand, saliency metrics used for pruning estimate the effect of removing parameters on the network's performance (measured by the accuracy or loss). Hence, saliency metrics for pruning have a measurable ground truth provided by observing how the network's performance changes when removing said parameters.

Xu et al. (2020) categorise estimation criteria into different groups depending on how they identify weights for pruning: importance-based, sparsity-based, and reconstruction-based. Importance-based saliency metrics are the most common type of saliency metrics. They try to gauge how important a parameter is to the network. Examples of importance-based metrics range from metrics that assume smaller weights have lesser importance (Han et al. 2015; Mao et al. 2017; Lebedev and Lempitsky 2016; Wang et al. 2018; Guo et al. 2016) to metrics that approximate the sensitivity of the network (Mozer and Smolensky 1988; LeCun et al. 1989; Hassibi and Stork 1992; Theis et al. 2018; Ding et al. 2019). Reconstruction-based saliency metrics find and prune redundant weights by reconstructing associated feature maps using the pruned weights (Luo et al. 2017; Luo et al. 2019; He et al. 2017; Liu et al. 2021).

Liu et al. (2020a) propose another classification for saliency metrics. They categorise saliency metrics depending on whether they use training data or not. Xu et al. (2020) categorise saliency metrics as data-agnostic or data-driven. Data-agnostic saliency metrics use only the permanent parameters, i.e. weights, of the network. In addition to the use of permanent parameters, data-driven saliency metrics use training information. The training information can either be training images only or training images with training labels. In this section, we use the classification given by Liu et al. (2020a) to classify saliency metrics. Our own proposed classification of saliency metrics is discussed in Chapter 3.

Data-agnostic Saliency Metrics

According to Liu et al. (2020a) data-agnostic, also referred to as weight-based, saliency metrics do not use training information. Commonly used data-agnostic saliency metrics are the L1-norm of the weights (Mao et al. 2017) and the mean squares of the weights(Molchanov et al. 2017). The L2 and L1 norm of the weights have been used in multiple pruning schemes (Lebedev and Lempitsky 2016; Li et al. 2017a; He et al. 2018a) for different granularities of pruning. Most weight-based saliency metrics assume that weights of lower magnitude have a lower contribution to the network. Yu et al. (2018) propose a recursive weight-based saliency metric to bound the error in the last layer of the network.

In Section 3.5, we discuss existing data-agnostic saliency metrics for channel pruning in CNNs in more details.

Data-driven Saliency Metrics

Data-driven saliency metrics can exploit the information in the feature maps and gradients (Liu et al. 2020a). These can respectively only be obtained by performing a forward, and an additional backward pass of the network.

Some examples of effective data-driven saliency metrics that use the feature maps are the average percentage of zeros (Hu et al. 2016), mean (Anwar et al. 2017), and standard deviation of feature maps (Polyak and Wolf 2015). Feature map-based saliency metrics exploit information only obtainable during forward passes of the network.

In Section 3.6, we discuss data-driven saliency metrics that use information obtainable from forward passes of the network.

Conversely, we can find saliency metrics that make use of only the gradients (Karnin 1990; Lee et al. 2019) such as the use of the average of the gradients, (Liu and Wu 2019). Saliency metrics that use the gradients exploit information availble only during backward passes of the network.

However, the information contained in the gradients is often coupled with the feature map points or the weights. The 2nd order Taylor expansion (Theis et al. 2018; Hassibi and Stork 1992; LeCun et al. 1989) and 1st order Taylor expansion

(Molchanov et al. 2017; Molchanov et al. 2019) are two notable examples of saliency metrics combining both the information of the feature maps and their gradients.

In Section 3.7, we discuss existing data-driven saliency metrics that use information from backward passes of the network.

2.5 Penalty Terms and Trainable Masks

2.5.1 Penalty Terms

The removal of weights from the network can be done using penalty terms (Reed 1993) in a gradual way. Pruning using penalty terms can be simplified to a modification of the network's cost function to promote networks with fewer weights. An example of the modified loss function, $\mathcal{L}_{new}$, using a simple L1-regulariser as a penalty term to favour smaller weights is given in Equation 2.5 (Reed 1993).

$$\mathcal{L}_{new} = \mathcal{L} + \|W\|_1 \tag{2.5}$$

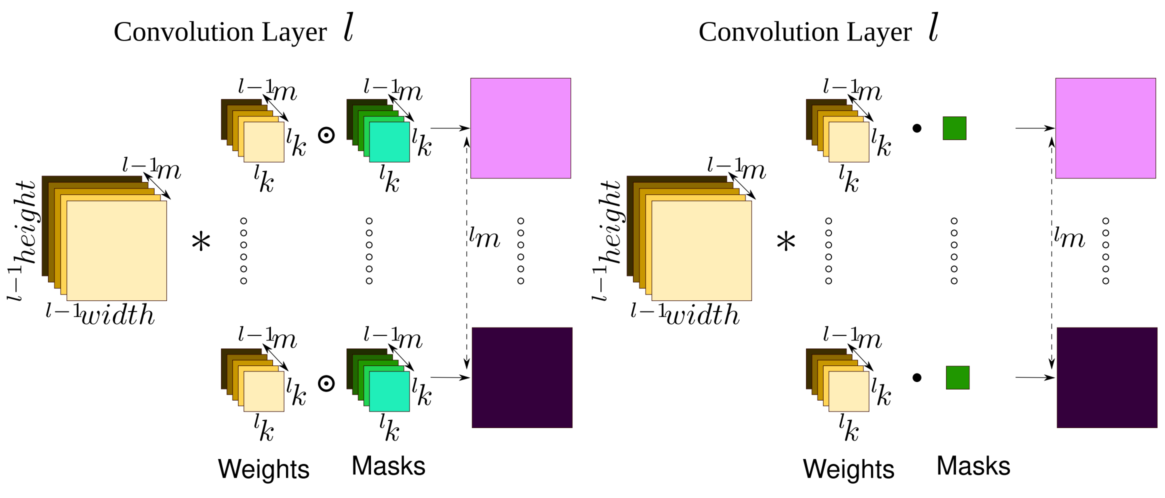
Metrics used for saliency-based pruning can also be used to estimate less important parameters and gradually penalise them (Lebedev and Lempitsky 2016; Cai et al. 2021).

2.5.2 Trainable Masks

Mask terms are parameters that can be used to turn on or off the participation of its associated weight or group of weights in the network. The structure of mask terms for individual weights and channel weights is shown in Figure 2-11.

Mask terms can be used to describe the pruning process without introducing new parameters to the network (Mozer and Smolensky 1988; Lee et al. 2019; Frankle and Carbin 2019). The use of masks terms in this way is merely to facilitate the description of pruning.

Trainable masks introduce new parameters into the optimisation process. For fine-grain pruning, each parameter has its own mask parameter (Ding et al. 2019). Masks for fine-grain pruning are of the same shape as the weight tensor. This structure is shown in Figure 2-11a. However, when gradually pruning structured pat-



(a) Structure of masks for fine-grain pruning. (b) Structure of masks for channel pruning.

Figure 2-11: Structures of masks for different granularities of pruning.

terns of the weight tensor, a common mask value is used for groups of weights that need to be pruned together (Yu et al. 2017; Liu et al. 2017b; Huang and Wang 2018). Hence, each pruning element has its own mask value and weights of a single pruning element share their mask value. This mask structure is shown in Figure 2-11b.

Binary masks only allow the mask values to be "1" or "0" to either allow the pruning element to participate in the classification task of the network or to be pruned (Yu et al. 2017; Liu et al. 2020b; Luo and Wu 2020).

Trainable masks that allow the mask terms to have continuous values between "0" and "1" can also be used to gradually prune weights (Kang and Han 2020; Ramakrishnan et al. 2020).

2.6 AutoML for Pruning

Automated machine learning (AutoML) has been used to solve many machine learning problems (Zoph and Le 2017; Liu et al. 2017a) including pruning (He et al. 2018b). He et al. (2018b) use a reinforcement learning agent to automatically subsample the pruning space and find the best layer-wise pruning rates. Liu et al. (2019a) treat channel pruning as a Network Architecture Search (NAS) problem and encode each layer structure with a vector. Liu et al. (2019a) use an evolutionary search algorithm to find pruned networks. Lin et al. (2020) apply ABC (Karaboga 2005) to pruning.

Chapter 3

Taxonomy of Saliency Metrics for Channel Pruning

A saliency metric estimates which parameters can be safely pruned with little impact on the classification accuracy of the DNN. Many saliency metrics have been proposed, each within the context of a wider pruning algorithm. The result is that it is difficult to separate the effectiveness of the saliency metric from the wider pruning algorithm that surrounds it. Similar looking saliency metrics can yield very different results because of apparently minor design choices.

In this chapter, we propose a taxonomy of saliency metrics based on four independent principal components: base input, pointwise metric, reduction method, and scaling method. We show that a broad range of metrics from the pruning literature can be grouped according to these components. Our taxonomy serves as a guide to prior work and allows us to construct new saliency metrics by exploring novel combinations of our taxonomic components. We also propose a novel scaling method based on the number of weights transitively removed.

We perform an in-depth experimental investigation of more than 300 saliency metrics made up of existing techniques and new combinations of components. We demonstrate the importance of reduction and scaling when pruning groups of weights. We find that some of our constructed metrics can outperform the best existing state-of-the-art metrics for convolutional neural network channel pruning. We also find that our novel scaling method improves existing saliency metrics.

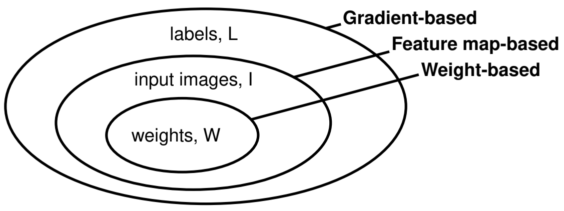


Figure 3-1: Saliency metrics can be grouped into three categories (weight-based, feature map-based, and gradient-based) depending on the information that they use. Weight-based metrics use weights of the network, feature map-based metrics use weights and input images, and gradient-based metrics use weights, input images and input labels to compute saliency. Examples of these different categories of metrics for channel pruning are given in Tables 3.2, 3.3, and 3.4 respectively.

3.1 Introduction

There is a huge variety of pruning algorithms in the literature, the vast majority have, at their heart, a saliency metric. A saliency metric is used to answer a fundamental question in pruning: which weight or set of weights, when removed, will likely cause the least damage to the network predictions? Since the saliency metric is typically presented within the context of a larger pruning algorithm, it is often extremely difficult to isolate the effect of the saliency metric from other design choices. In this chapter, we evaluate a wide range of existing and novel saliency metrics within the same pruning algorithm.

3.1.1 Contributions

In this chapter, we study the impact of the choice of saliency metric within a single canonical channel pruning algorithm for CNNs. Although our empirical results are for this specific context, there is a strong argument for the generality of our findings, which we highlight in discussion. We make the following specific contributions.

- We propose a taxonomy that classifies saliency metrics based on four independent components and classify existing saliency metrics using the proposed taxonomy. Our taxonomy allows us to identify the common elements and differences between saliency metrics that may be presented as very different in the literature.

- We present the first large-scale empirical evaluation of 308 saliency metrics, including metrics from prior work and novel metrics derived from new combinations of taxonomic components.
- We experimentally confirm a widely-acknowledged rule of thumb: gradient-based approaches as a class significantly outperform simpler weight-based methods.
- We answer long-standing open research questions, such as whether the popular strategy of ignoring first-order terms in Taylor-expansions is safe in practice.
- We find the previously unknown result that the choice of dimensionality reduction and parameter scaling method has a large impact on saliency metrics.
- We propose a novel scaling method based on transitive elimination of weights in channel pruning, which significantly outperforms the best existing scaling methods.
- We show that good saliency metrics can be effective even without any subsequent retraining, or greatly reduce the number of iterations required if retraining is used.

3.2 Background

We propose a grouping of salience metrics based on the information used to compute them. Figure 3-1 groups saliency metrics according to the information that they use. Saliency metrics that use only the weights have the advantage of having all required information readily available. Data driven approaches require training data to make pruning decisions. Approaches that only use input images to make pruning decisions require only forward passes of the network. Approaches that use gradients additionally require backward passes of the network to compute the loss with respect to input labels and hence the gradients. The main differentiating factor between these classes of approach (Figure 3-1) is in practice the cost associated with the use of more information.

Saliency metrics can also be grouped according to how they identify least important weights (Xu et al. 2020). Simple metrics like the L1-norm of weight (Han et al. 2015) or APoZ (Hu et al. 2016) assume that small weights or feature maps with high frequencies of zeros are less important to the network. Taylor expansion-based metrics often approximate the change in global (LeCun et al. 1989; Hassibi and Stork 1992; Molchanov et al. 2017; Ding et al. 2019) or layer-wise (Dong et al. 2017a) loss caused by pruning and remove weights that cause the least change in loss. Luo et al. (2019), He et al. (2017), and Liu et al. (2021) choose weights that lead to the least feature map reconstruction error. Hur and Kang (2019) use the entropy of the weights to determine which weights are least important.

While these saliency metrics are derived with different assumptions, if they can be expressed in a standard form then finding the common elements and differences becomes easier. We can also compare different metrics more easily.

3.3 A Taxonomy of Saliency Metrics

We propose a taxonomy of saliency metrics based on four principle components. A fine-grain saliency metric is constructed from a pointwise metric F over the parameter set X. When we prune larger groups of weights, such as entire channels,

Base Input	**Pointwise Metric** $f(x)$	**Reduction** R	**Scaling** K
$X = W$	x	$\sum_{x \in {}^l X_i} x$	1
$X = A$	$\frac{d\mathcal{L}}{dx}$	$\sum_{x \in {}^l X_i} \lvert x \rvert$	$n({}^l X_i)$
	$-x\frac{d\mathcal{L}}{dx}$	$\left\lvert \sum_{x \in {}^l X_i} x \right\rvert$	$\left\lVert {}^l \widetilde{S} \right\rVert_1$
	$-x\frac{d\mathcal{L}}{dx} + \sum_{y \notin \tilde{W}} \frac{xy}{2} \frac{d^2\mathcal{L}}{dxdy}$	$\sum_{x \in {}^l X_i} (x)^2$	$\left\lVert {}^l \widetilde{S} \right\rVert_2$
	$x - reconstruction(x)$	$\left(\sum_{x \in {}^l X_i} x \right)^2$	$n(\mathcal{TC}({}^l W_i))$
	$alternateBackprop(x)$	$\sqrt{\sum_{x \in {}^l X_i} (x)^2}$	

Table 3.1: A taxonomy of published channel saliency metrics. One component from each column is chosen to construct a channel saliency metric.

we need to combine the saliency of individual weights into a metric for the entire channel. We do this with a dimensionality reduction R and normalisation K. Some examples of choices for X, F, R and K are given in Table 3.1.

The general form of the saliency metric for an arbitrary subset of parameters X is given by Equation 3.1.

$$S = \frac{1}{K} \cdot \widetilde{S}, \text{ with } \widetilde{S} = R \circ F(X) \tag{3.1}$$

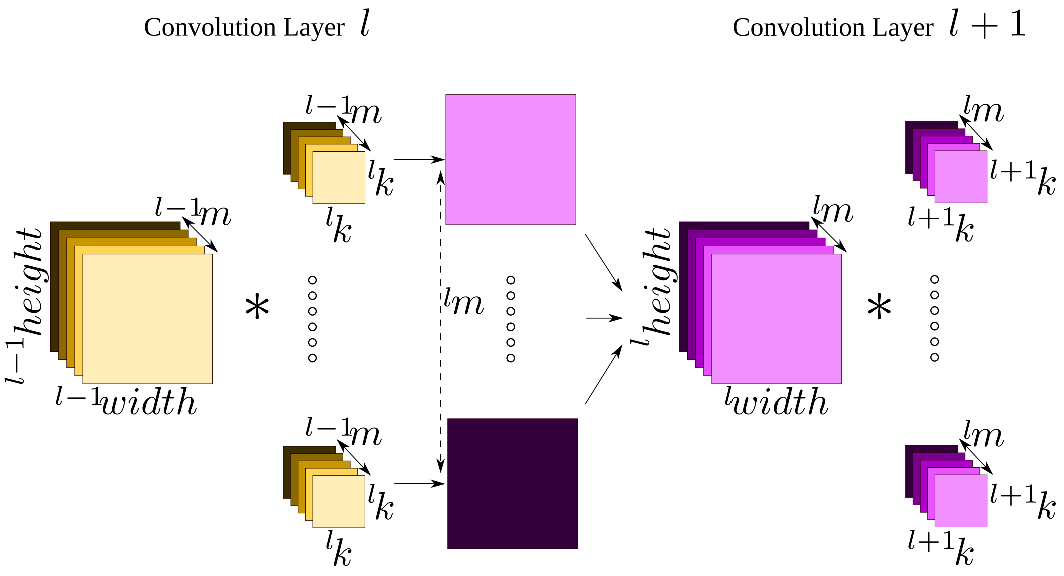


Figure 3-2: The complete description of our notation is described in Section 2.2.1

In our mathematical treatment, we consider the general case with W unless stated otherwise. However, in our experimental evaluation, we are concerned with parameter subsets X corresponding to the parameters ${}^{l}W_i$ of a channel of a convolutional layer. Thus, ${}^{l}S_i$ denotes the saliency of the i^{th} channel of the l^{th} convolution layer of the CNN.

To facilitate the description and comparison of different saliency metrics, we use the notation introduced in Section 2.2.1 and summarised in Figure 3-2.

3.3.1 Domain (Choice of X)

We propose a taxonomy where saliency metrics are based upon the weights themselves ($X = W$), or the values of output features maps that are computed using the weights ($X = A$). There is often a close relationship between the magnitude of a weight and the sensitivity of the DNN to pruning the weight, so many saliency metrics use the weight as a key input. However, in the specific case of pruning entire output channels, there is a direct relationship with the output points corresponding to the i^{th} channel of the l^{th} convolution layer ${}^{l}C_i$ (i.e. the feature map ${}^{l}A_i$). Removing all parameters contributing to an output feature map in a convolutional layer results in the feature map becoming zero. When this happens, all of the operations which are transitively used to compute the operation can also be pruned, resulting in large savings.

Hence, the saliency of a channel can be regarded as a function of outputs (Hu et al. 2016; Gaikwad and El-Sharkawy 2019; Anwar et al. 2017; Polyak and Wolf 2015), rather than parameters, i.e. in Equation 3.1 X can be either the weights, ${}^{l}W_i$, or the output feature map, ${}^{l}A_i$. The relationship between the weights and feature maps of a channel is illustrated in details in Figure 3-3. The removal weights of the output channel in layer l (highlighted in red) directly maps to the removal of its corresponding feature map (highlighted in blue). Since output feature maps are ultimately used by the following layer, weights from the following layer can also be removed (highlighted in green). This relationship is further explained in Section 5.3.6.

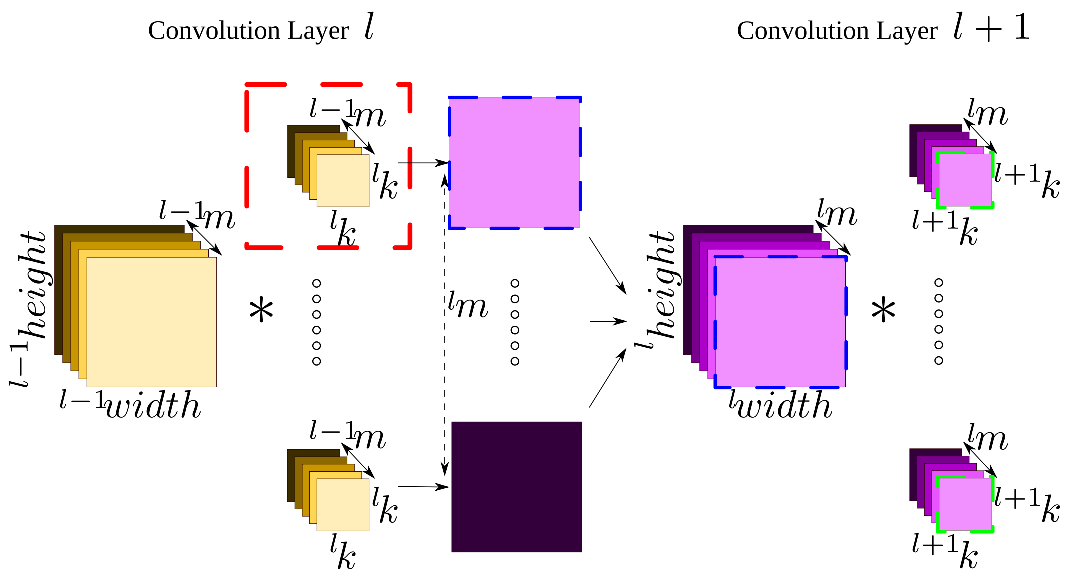


Figure 3-3: Parameter/Feature map structure in a convolutional neural network.

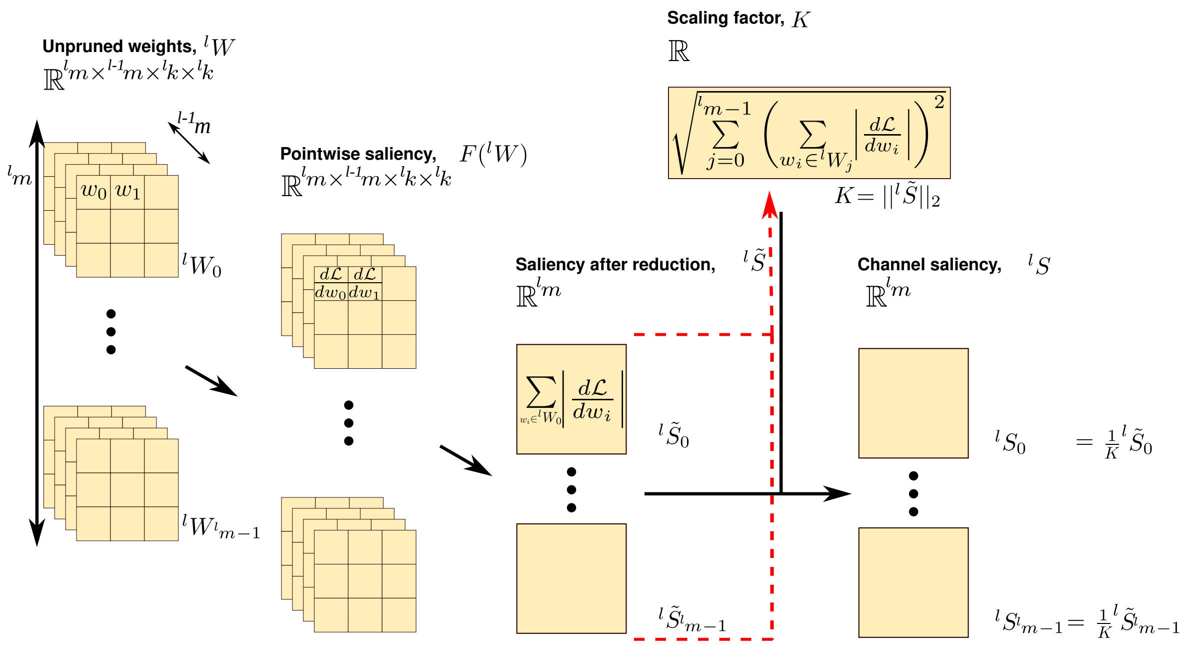
It should be noted that since channel pruning can be viewed as feature map removal, feature selection metrics can also be used as saliency metrics for pruning (Tang et al. 2020).

When pruning at granularities finer than entire channels, the removal of output points cannot be directly mapped to the removal of sets of weights. Hence, it is clear that $X = W$ is the best choice. However, for channel pruning it is difficult to choose definitively between output feature map or weights. In fact, most published saliency metrics are presented either as functions of weights or of output points, despite being applicable to both. Some metrics (LeCun et al. 1989; Hassibi and Stork 1992) were originally defined using weights (as they were used to prune individual weights), but derived metrics (Molchanov et al. 2017; Theis et al. 2018) use output feature maps instead. Thus, there is a great deal of disagreement in the existing literature about whether $X = W$ or $X = A$ is the better choice. In Section 3.9, we experimentally evaluate the two choices.

To better illustrate the effect of the four independent choices, we use an example for channel pruning. In Figure 3-4, we show how to compute the saliency of a convolution layer's channels using its weights. This corresponds to the case where $X = W$.

3.3.2 Pointwise Metric (Choice of F)

We denote $F(X)$ the tensor of pointwise saliency of all individual weights or output points. $F(X)$ is of the same shape as X, that is, either of the shape of W or A. When pruning, it is common to look at either the saliency of an individual element of the saliency vector or at a group of them. To facilitate this grouping, we introduce ${}^{l}F(X)_i$ and $f(x)$. ${}^{l}F(X)_i$ is the tensor of saliency corresponding to the i^{th} channel of the l^{th} layer. ${}^{l}F(X)_i$ is of the same shape as ${}^{l}X_i$. Hence if $X = W$ or $X = A$, then ${}^{l}F(X)_i$ is of the shape ${}^{l-1}m \times {}^{l}k \times {}^{l}k$ or ${}^{l}height \times {}^{l}width$ respectively. $f(x)$, is used to denote the saliency of a single weight or output point. If $x = {}^{l}W_i[p, q, r]$ is an individual weight from ${}^{l}W_i$, then $f(x) = {}^{l}F(X)_i[p, q, r]$. Similarly, when using the output feature map instead of the weights, if $x = {}^{l}A_i[p, q]$ is an individual output point then $f(x) = {}^{l}F(X)_i[p, q]$.



Unpruned weights, ${}^{l}W$
$\mathbb{R}^{{}^{l}m \times {}^{l\text{-}1}m \times {}^{l}k \times {}^{l}k}$
${}^{l\text{-}1}m$
${}^{l}m$
w_0 w_1
${}^{l}W_0$
${}^{l}W_{{}^{l}m-1}$
Pointwise saliency, $F({}^{l}W)$
$\mathbb{R}^{{}^{l}m \times {}^{l\text{-}1}m \times {}^{l}k \times {}^{l}k}$
$\frac{d\mathcal{L}}{dw_0}$ $\frac{d\mathcal{L}}{dw_1}$
Scaling factor, K
$\mathbb{R}$
$\sqrt{\sum_{j=0}^{{}^{l}m-1}\left(\sum_{w_i \in {}^{l}W_j}\left|\frac{d\mathcal{L}}{dw_i}\right|\right)^2}$
$K = \|{}^{l}\tilde{S}\|_2$
Saliency after reduction, ${}^{l}\tilde{S}$
$\mathbb{R}^{{}^{l}m}$
$\sum_{w_i \in {}^{l}W_0}\left|\frac{d\mathcal{L}}{dw_i}\right|$
${}^{l}\tilde{S}_0$
${}^{l}\tilde{S}_{{}^{l}m-1}$
Channel saliency, ${}^{l}S$
$\mathbb{R}^{{}^{l}m}$
${}^{l}S_0 \quad = \frac{1}{K}{}^{l}\tilde{S}_0$
${}^{l}S_{{}^{l}m-1} = \frac{1}{K}{}^{l}\tilde{S}_{{}^{l}m-1}$

Figure 3-4: Computing channel saliency using Equation 3.1 using $X = W$, $f(x) = \frac{d\mathcal{L}}{dx}$, $R(X) = \sum_{x \in {}^{l}X_i} |x|$ and $K = \left\|{}^{l}\tilde{S}\right\|_2$.

A common pointwise saliency function is the absolute magnitude function, i.e. the saliency of an individual weight or output point is given directly by its absolute value, hence $f(x) = |x|$.

In Figure 3-4, the gradient of the loss with respect to an element is used as a saliency metric, $F(X) = J$ or $f(x) = \frac{d\mathcal{L}}{dx}$. Applying F to W yields a tensor containing the saliency of the individual weights.

3.3.3 Dimensionality Reduction (Choice of R)

Once the pointwise saliency vector is obtained, a reduction is used to condense the tensor of pointwise saliency to a single value for the pruning element. In the case of channel pruning, R reduces either a ${}^{l-1}m \times {}^{l}k \times {}^{l}k$ tensor or a ${}^{l}height \times {}^{l}width$ tensor into a single value.

Any suitable vector norm could be used as a reduction, with the L2-norm being a popular reduction method in the literature (He et al. 2018a; Gaikwad and El-Sharkawy 2019).

In Figure 3-4, an L1-norm is used to reduce the ${}^{l}m \times {}^{l}k \times {}^{l}k$ tensor of weight saliency into a vector containing the layer's channel saliency , ${}^{l}\tilde{S}$, of dimension ${}^{l}m$.

3.3.4 Scaling (Choice of K)

All other things being equal, the more parameters which can be pruned, the lower the computational and memory costs of inference. Therefore, if two channels have similar saliency, one should favour pruning the larger channel, which is the Pareto-optimal choice considering the twin objectives of minimising accuracy loss and maximising the number of parameters pruned.

To better integrate this cost function, a scaling coefficient, K is typically used. One solution is to scale the channel saliency ${}^{l}S_i$ using the cardinality of the pointwise saliency vector. In other words, one can also look at the average saliency of a group instead of the sum of the saliency in the group. For example, instead of using the sum of the magnitudes or L1-norm of the weights, $\left\|{}^{l}W_i\right\|_1$, one can use the average of the magnitudes, $\frac{1}{n({}^{l}W_i)}\left\|{}^{l}W_i\right\|_1$. This normalises the result of the reduction, R, so that channels with many weights are more likely to be pruned, leading to greater overall sparsity. Another solution is to perform a layer-wise normalisation to scale the magnitudes of saliency across layers. Using a layer-wise L2-norm in the case of global pruning helps when values for saliency in different layers have drastically different magnitudes (Molchanov et al. 2017).

In Figure 3-4, we use a layer-wise L2-norm as scaling factor. We use the L2-norm of the unscaled saliency of all the channels in the given convolution layer, ${}^{l}\widetilde{S}$, to obtain the scaling coefficient, K, of that layer.

Proposed Scaling Method, $\mathcal{TC}$

We propose a novel scaling method $\mathcal{TC}$ that has not been considered by previous researchers. $\mathcal{TC}({}^{l}W_i)$ is used to denote the entire set of weights transitively removed when ${}^{l}W_i$ is removed from the network. A simple example of $\mathcal{TC}({}^{l}W_i)$ is shown in Figure 3-3, where weights from the next convolution layer are also removed when we remove an output channel from the network. Since $\mathcal{TC}({}^{l}W_i)$ (highlighted in red or green) include ${}^{l}W_i$ (highlighted in red), $n(\mathcal{TC}({}^{l}W_i)) \geq n({}^{l}W_i)$. A more detailed explanation of TC is given in Section 5.3.1. When optimising for the maximum number of weights removed for the least loss in accuracy, $\mathcal{TC}({}^{l}W_i)$ is interesting because it takes into account all the weights removed for that channel.

3.3.5 Minibatches

Data driven approaches which consider gradients or output points in the domain X often rely on a set of inputs to produce these values. Since each input in the set will result in a potentially different set of saliency values, the computation of the saliency over a minibatch is typically done by combining the element-wise saliency values using a simple average across the minibatch (Theis et al. 2018; Molchanov et al. 2017; Gaikwad and El-Sharkawy 2019; Hu et al. 2016).

Hence, the full saliency equation used in practice is given by Equation 3.2 with N, the total number of images used in the minibatch.

$$^{l}S_i = \frac{1}{N}\sum_{n=0}^{N-1}\frac{1}{K}\cdot R \circ F(X_n) \tag{3.2}$$

In some cases, a square root is applied to the resulting saliency metric (Gaikwad and El-Sharkawy 2019). This additional computation does not modify the ranking of the channels and can thus be omitted for algorithms only concerned with channel ranking.

3.4 Classification of Existing Saliency Metrics

In this section, we classify popular saliency metrics using our taxonomy. Tables 3.2, 3.3, and 3.4 summarise channel saliency metrics that have been used for pruning convolution channels. We can obtain new saliency metrics by selecting different combinations of the four components in Table 3.1. Although some of these saliency metrics resemble each other, their efficacy can vary.

This thesis focuses on using saliency metrics specifically for channel removed together. However, the taxonomy can also be used to classify saliency metrics used for different granularities. For fine-grain pruning, i.e. pruning individual weights, only the pointwise saliency is relevant. For other granularities of pruning, pointwise metrics are computed across the relevant substructure in parameter tensors and then dimensionally reduced (e.g. with a vector or tensor norm) to yield structural metrics. Table 3.5 shows the classification of a selection of popular pointwise saliency metrics that have been used for fine-grain pruning.

Method	**Base Input,** X	**Pointwise Measure,** $f(x)$	**After Reduction,** ${}^l\widetilde{S}_i$	**Scaling,** K
Using only weights				
L1-norm of weights (Mao et al. 2017; Li et al. 2017a; Wang et al. 2018)	W	x	$\sum_{x\in i\,{}^lX_i} \|f(x)\|$	1
L2-norm of weights (Lebedev and Lempitsky 2016; He et al. 2018a)	W	x	$\sum_{x\in {}^lX_i} f(x)^2$	1
Min-weight (Molchanov et al. 2017)	W	x	$\sum_{x\in {}^lX_i} f(x)^2$	$n({}^lX_i)$
NISP (Yu et al. 2018)	W	$alternateBackprop(x)$ with $alternateBackprop(x) = NISP(x)$ (Equation 3.9)	$\sum_{x\in {}^lX_i} f(x)$	1
Geometric median of weights (He et al. 2019)	W	$x - reconstruction(x)$ with $reconstruction(x) = GM(x)$ (Equation 3.5)	$\sum_{x\in {}^lX_i} f(x)^2$	1

Table 3.2: Published approaches for channel pruning using weights only.

Method	**Base Input,** X	**Pointwise Measure,** $f(x)$	**After Reduction,** ${}^{l}\widetilde{S}_i$	**Scaling,** K
Using weights and input images				
Sum of feature map (Anwar et al. 2017)	A	x	$\sum_{x \in {}^{l}X_i} f(x)$	1
APoZ (Hu et al. 2016)	A	$\begin{cases} 1, \text{if } x > 0 \\ 0, \text{else} \end{cases}$	$\sum_{x \in {}^{l}X_i} f(x)$	$n({}^{l}X_i)$
L2-norm of activations (Gaikwad and El-Sharkawy 2019)	A	x	$\sum_{x \in {}^{l}X_i} f(x)^2$	1

Table 3.3: Published approaches for channel pruning using weights and input images.

Method	**Base Input,** X	**Pointwise Measure,** $f(x)$	**After Reduction,** ${}^l\widetilde{S}_i$	**Scaling,** K
Using weights, input images and labels				
Fisher information using activations (Theis et al. 2018; Turner et al. 2018)	A	$x\frac{d\mathcal{L}}{dx}$	$\left(\sum\limits_{x\in{}^lX_i} f(x)\right)^2$	$\frac{1}{2}$
Fisher information using weights (Molchanov et al. 2019)	W	$x\frac{d\mathcal{L}}{dx}$	$\left(\sum\limits_{x\in{}^lX_i} f(x)\right)^2$	1
1st Order Taylor (Molchanov et al. 2017)	A	$x\frac{d\mathcal{L}}{dx}$	$\left\lvert\sum\limits_{x\in{}^lX_i} f(x)\right\rvert$	$n({}^lX_i)$
1st Order Taylor, w. norm (Molchanov et al. 2017)	A	$x\frac{d\mathcal{L}}{dx}$	$\left\lvert\sum\limits_{x\in{}^lX_i} f(x)\right\rvert$	$\left\lVert{}^l\widetilde{S}\right\rVert_2$
Average of gradient (Liu and Wu 2019)	A	$\frac{d\mathcal{L}}{dx}$	$\sum\limits_{x\in{}^lX_i} f(x)$	$n({}^lX_i)$
Collaborative channel pruning (Peng et al. 2019)	W	$\frac{1}{2}x_0x_1\frac{d^2\mathcal{L}}{dx_0dx_1}$ (Table 3.6)	$\sum\limits_{\substack{x_0,x_1:\\ x_0,x_1\notin\widetilde{{}^lX_i}}} f(x_0,x_1)$	1
Connection sensitivity (Lee et al. 2019)	W (Section 3.7.1)	$x\frac{d\mathcal{L}}{dx}$	$\sum\limits_{x\in{}^lX_i} \lvert f(x)\rvert$	$\sum\limits_{s\in{}^l\widetilde{S}} s$

Table 3.4: Published approaches for channel pruning using weights, input images and labels.

Method	**Pointwise Measure,** $f(x)$
Magnitude (Han et al. 2015; Han et al. 2017; He et al. 2018b; Frankle and Carbin 2019; Guo et al. 2016)	$\lvert w \rvert$
Optimal Brain Damage (LeCun et al. 1989)	$\frac{d^2\mathcal{L}}{dw^2}$
Optimal Brain Surgeon (Hassibi and Stork 1992)	$\frac{w_i w_j}{2}\frac{d^2\mathcal{L}}{dw_i dw_j}$
Mask gradient (Mozer and Smolensky 1988)	$-\frac{d\mathcal{L}}{dm}$
Weight gradient (Karnin 1990)	$-\frac{d\mathcal{L}}{dw}$
1st order Taylor expansion (Ding et al. 2019)	$\left\lvert \frac{d\mathcal{L}}{dw} w \right\rvert$

Table 3.5: Published approaches for fine-grain pruning.

3.5 Weight-based Saliency Metrics

When computing weight-based metrics, all the information required to compute the saliency is readily available. The most common saliency metric used for pruning is the magnitude of the weights. Specifically, the L2 and L1 norms of weights have been used in many pruning algorithms (Mao et al. 2017; Lebedev and Lempitsky 2016; Li et al. 2017a; He et al. 2018a; Wang et al. 2018; Han et al. 2015; Han et al. 2017; He et al. 2018b; Frankle and Carbin 2019; Guo et al. 2016) for different granularities of pruning. Magnitude-based saliency metrics assume that weights of lower magnitude have a lesser contribution to the network.

Another weight-based heuristic is *min-weight* (Molchanov et al. 2017). The sum of squared individual weights are scaled by the number of weights in the channel to give the channel saliency (Equation 3.3).

$$ {}^{l}S_i = \frac{1}{n({}^{l}W_i)} \sum_{w \in {}^{l}W_i} w^2 \tag{3.3} $$

Most weight-based metrics consider smaller weights to be less important to the network. For coarse granularities of pruning, one can also consider removing redundant sets of weights. In the case of channel pruning, one can remove channels that are similar to other channels. Redundant channels can be removed if they are not contributing to the final result. (He et al. 2019) remove channels that have a Euclidean distance close to that of the layer′s geometric median channel (see Equation 3.4).

$$^{l}W_{GM} \in argmin(g(x)) \text{ with } g(x) = \sum_{j=0}^{^{l}m-1} \left\|x - {}^{l}W_j\right\|_2 \tag{3.4}$$

In this case, the pointwise saliency $f(x)$ is given by $x - GM(x)$ where $GM(x)$ is given according to Equation 3.5.

$$\text{Given } x = {}^{l}W_i[p,q,r], \text{ then } GM(x) = {}^{l}\mathcal{W}_{GM}[p,q,r] \tag{3.5}$$

By removing channels that are close to the geometric median, one can assume that they are already represented by the geometric median. We denote $reconstruction(x)$ any reconstruction of x after pruning, then in this case, $reconstruction(x) = GM(x)$.

3.5.1 Recursive Weight-based Metrics

Common weight-based methods only use weights from a single layer. Saliency metrics such as the L1-norm of weights treat channels as independent components.

To illustrate this independence, let us consider the example of using the L1-norm of weights as a saliency metric. First, we compute the saliency metric using the L1-norm of weights then remove the least salient channel. We then recompute the saliency of the remaining weights. The saliency of the channels that were not pruned remain unchanged.

Now, let us consider the case where we use the L1-norm of output points. We compute the saliency of all the channels, prune the least salient channel and finally recompute the saliency of the unpruned channels. Let us assume that the channel selected for pruning is from the third layer of the network. The saliency of the channels from the first, second, and third layers (excluding the pruned channel) are unchanged. However, the saliency of channels from following layers may have changed. While the saliency of the channels are computed in an independent way, the underlying information (the output points), can be expressed recursively using the previous layers. The recursive component is implicit. The recursive equation for outputs points is given in Equation 3.6 with f^l being the forward pass function of the l^{th} layer.

$$ {}^{l}A = f^{l}({}^{l}W, {}^{l-1}A) \tag{3.6} $$

Saliency metrics that use the gradients of the loss with respect to the weights or the output points also have a component that is computed recursively using back-propagation (see Equation 3.7). Hence, the pointwise metrics in Table 3.1 can use information from different layers.

$$ \frac{d\mathcal{L}}{d\,{}^{l}A} = \frac{d\mathcal{L}}{d\,{}^{l+1}A}\frac{d\,{}^{l+1}A}{d\,{}^{l}A} \tag{3.7} $$

Neuron Importance Score Propagation (NISP) (Yu et al. 2018) uses an explicitly recursive way of propagating saliency information between layers.

Equation 3.8 shows the general propagation equation used in NISP with h^{l} being a function given by the authors. f^{l} depends only on the type of the layer.

$$ {}^{l}\mathcal{S} = h^{l}({}^{l+1}W, {}^{l+1}\mathcal{S}) \tag{3.8} $$

In this case, the pointwise saliency $f(x)$ used by $NISP(x)$ is given in Equation 3.9.

$$ \text{Given } x = {}^{l}W_{i}[p,q,r], \text{ then } NISP(x) = {}^{l}\mathcal{S}[p,q,r] \tag{3.9} $$

The method used by NISP can be considered as an alternative way of backpropagating information through the network.

Gradient backpropagation and its alternatives can be used for pruning. An alternative to backpropagation of gradients is Layer-wise Relevance Propagation (LRP) (Bach et al. 2015). LRP has successfully been used as a saliency metric for pruning (Yeom et al. 2021). Similar to gradient backpropagation and NISP, LRP propagates information recursively from the last layer (output) of the network to its first layer (input). LRP requires weights and input images as it is a data driven approach but can, nonetheless, be expressed in a similar form to Equation 3.8.

We denote $alternateBackprop$ any alternative to backpropagation of information in neural networks. Hence, in the case of NISP, we have $alternateBackprop(x) = NISP(x)$.

3.6 Weight and Input Images-based Saliency Metrics

Channel pruning is a notable granularity of pruning as pruning an entire channel of weights leads to the removal of a feature map from the network. A given channel of weights that operates on a given input produces an output feature map of outputs. It may be possible to identify good candidates for pruning by selecting feature maps with low or zero outputs.

Output values across inferences on multiple inputs can be gathered, and their results summarised using statistical measures. Saliency metrics use the sum (Anwar et al. 2017), mean, and variance (Polyak and Wolf 2015), or L2-norm (Gaikwad and El-Sharkawy 2019) of feature maps to identify low saliency channels.

The feature map produced by the convolution layer is not the only feature map that can be used. For example, the absolute percentage of zeros (APoZ) (Hu et al. 2016) counts the percentage of zero values in the output feature map for a given channel and computes the average across multiple inputs.

$$ {}^{l}S_i = \frac{1}{n({}^{l}A_i)} \sum_{a \in {}^{l}A_i} a \tag{3.10} $$

Quite often, convolution layers are followed by ReLU layers which only retain positive outputs. A negative mean would indicate that on average the outputs produced by that channel were negative and likely to be driven to zero by ReLU. Hence, APoZ considers the average of the output points after ReLU. APoZ considers channels with a higher percentage of zeros to have a lower saliency.

$$ {}^{l}S_i = \frac{1}{n({}^{l}A_i)} \left(\sum_{a \in {}^{l}A_i} f(a) \right) \text{with} f(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{else} \end{cases} \tag{3.11} $$

A sub-category of saliency metrics that use weights and input images are metrics inspired from reconstruction error. Saliency metrics that are based on reconstruction error have a pointwise metric in the form of $x - reconstruction(x)$. In the case of ThiNet (Luo et al. 2019), x is a point sampled from the input feature map and $reconstruction(x)$ is its reconstruction after pruning. Metrics used by Luo et al. (2019), He et al. (2017), and Liu et al. (2021) choose channels based on the least error incurred to output feature maps. Hence, the layer-wise feature maps error is used as a saliency metric. The main difference between the approach explored by Luo et al. (2019) and He et al. (2017) is how they estimate the damaged feature map. Liu et al. (2021) also introduce a layer-wise loss alongside the reconstruction error.

3.7 Weight, Input Images, and Labels-based Saliency Metrics

Using only the weights and input images to compute saliency metrics does not allow one to know directly how the classification performance is being affected. Corresponding labels are also needed to obtain this information. The loss is a measure of how well the predictions of the network match the ground truth. Consequently, the gradients with respect to the loss also carry this information. A network that has reached its minimum loss has a gradient equal to zero.

The equation of the cross-entropy loss, $\mathcal{L}$, is given by Equation 2.1 for a network with weights W and input dataset $\mathcal{I}_{set}$. We remind the definition of the loss function from Section 2.2.1 in Equation 3.12.

$$\mathcal{L}(W, \mathcal{I}_{set}) = \sum_{n=0}^{N-1} {}_n\mathcal{L}(W) = \sum_{n=0}^{N-1} -({}_nL \odot log(P(W, {}_nI))) \tag{3.12}$$

P is evaluated during a forward pass of the network using only the weights and input images. On the other hand, evaluating $\mathcal{L}$ requires the ground truth and so do the gradients of the loss. Hence, the use of the $loss$ and its gradients carry more information than using only the weights or output feature maps. To compute the gradient of the loss, a backward pass as well as a forward pass is required.

The use of gradients in saliency metrics for pruning was introduced by Mozer and Smolensky's Skeletonization (Mozer and Smolensky 1988), Lecun et al.'s Optimal Brain Damage (LeCun et al. 1989), and Hassibi and Stork's Optimal Brain Surgeon (Hassibi and Stork 1992).

A simpler gradient-based saliency measure was proposed by Liu and Wu (2019) where the average of the output feature map gradients (Equation 3.13) is used as a saliency measure of a channel. They put forward that pruning channels that are no longer updated by the SGD algorithm can be pruned safely.

$$ {}^{l}S_i = \frac{1}{n({}^{l}A_i)} \sum_{a \in {}^{l}A_i} \frac{d\mathcal{L}}{da} \tag{3.13} $$

While Optimal Brain Damage introduced the use of Taylor expansions for deriving saliency methods, other more recent approaches have also used Taylor expansions to obtain different saliency metrics. These methods are further explained in Section 3.7.2

Saliency metrics that use a Taylor expansion estimate the error caused to the final loss of the network. Dong et al. (2017a) introduce a layer-wise error, hence a layer-wise sensitivity and propose a method to propagate this layer-wise sensitivity to deduce the final impact on the network. They use the saliency measure introduced by Optimal Brain Surgeon (Hassibi and Stork 1992) to estimate the layer-wise sensitivity.

3.7.1 Connection Sensitivity

Lee et al. (2019) define a saliency measure, Connection Sensitivity, based on gradients of the loss with respect to mask terms. Each individual weight, w_i, have a mask term, m_i , that can be either one or zero. Their saliency measure is derived using the mask term gradients instead of the weight gradients.

$$ S(w_i) = \frac{|g_i(w;\mathcal{I}_{set})|}{\sum\limits_{k=1}^{m} |g_k(w;\mathcal{I}_{set})|} \tag{3.14} $$

$$ g_i(w;\mathcal{I}_{set}) = \frac{\partial \mathcal{L}(M \odot W;\mathcal{I}_{set})}{\partial m_i}\Big|_{M=1} \tag{3.15} $$

To facilitate the comparison of Connection Sensitivity to other saliency metrics, we remind a few notations. We use W to denote the vector of the weights and M, the vector of the mask terms. M and W are of similar dimensions. The vector of pruned weights, $\widetilde{W}$ with i^{th} element $\widetilde{w}_i$, is given by applying the mask on the weights with $\widetilde{W} = M \odot W$ or $\widetilde{w}_i = m_i \cdot w_i$

Using this substitution, in Equation 3.15, we obtain Equation 3.16. The gradient of $\mathcal{L}$ with respect to the $\widetilde{W}$, $\frac{d\mathcal{L}}{d\widetilde{W}}$, is given by regular backpropagation rules.

In the case of Connection Sensitivity (Lee et al. 2019), since $M = 1$, i.e. all the components of M are set to 1, we can express the mask term gradients in terms of the known weight gradients. From Equation 3.18, we see that in this case using the absolute value of the mask gradient, $|g_i|$, is similar to using the absolute value of the first term of a Taylor expansion, $\left|w_i \frac{d\mathcal{L}}{dw_i}\right|$, from Equation 3.22

$$\left.\frac{\partial \mathcal{L}(M \odot W; \mathcal{I}_{set})}{\partial m_i}\right|_{M=1} = \left.\frac{\partial \mathcal{L}(M \odot W; \mathcal{I}_{set})}{\partial \widetilde{w}_i} \cdot w_i\right|_{M=1} \tag{3.16}$$

$$= \frac{\partial \mathcal{L}(W; \mathcal{I}_{set})}{\partial \widetilde{w}_i} \cdot \widetilde{w}_i \tag{3.17}$$

$$= w_i \cdot \frac{\partial \mathcal{L}(W; \mathcal{I}_{set})}{\partial w_i} \tag{3.18}$$

3.7.2 Taylor Expansion

Most of the gradient-based saliency metrics discussed in this chapter can be summarised as the estimation of the sensitivity of the parameters removed by removing a convolutional filter channel from a network. The sensitivity of a parameter was first introduced for fully-connected layers (Mozer and Smolensky 1988) as the change in the error of the network on the training set caused by removing that parameter. This definition can be extended to convolution channels by using the change in error induced by removing the set of parameters associated with that channel. The sensitivity of pruning a network with loss, $\mathcal{L}$, and unpruned weights, W, to pruned weights, $\widetilde{W}$, is given in Equation 3.19.

$$Sensitivity = \mathcal{L}(\widetilde{W}) - \mathcal{L}(W) \tag{3.19}$$

One of the first approaches to estimate the sensitivity of a weight was proposed by Lecun et al.'s Optimal Brain Damage (LeCun et al. 1989). They use a simplified second order Taylor expansion on the trained neural network. A Taylor expansion is used to estimate the loss function, $\mathcal{L}$, at the pruned weights, $\widetilde{W}$, using the trained weights, W.

$$\begin{aligned}\mathcal{L}(\widetilde{W}) \approx \mathcal{L}(W) &+ J(\widetilde{W} - W) \\ &+ \frac{1}{2}(\widetilde{W} - W)^T H(\widetilde{W} - W)\end{aligned} \tag{3.20}$$

A second order Taylor expansion around the trained weights, W, is given in equation 3.20, where J and H are respectively the Jacobian and Hessian of the loss function at trained parameters W. The Jacobian matrix, $J \in \mathcal{R}^{N_{weights}}$, is defined as $J_i = \frac{d\mathcal{L}}{dwi}$ and the Hessian matrix, $H \in \mathcal{R}^{N_{weights} \times N_{weights}}$, is defined as $H_{ij} = \frac{d^2\mathcal{L}}{dw_i dw_j}$

To prune an individual weight, w_i, from the network, w_i is set to zero, i.e. the i^{th} parameter of $\widetilde{W}$ is set to zero. To easily understand how the saliency of a single parameter is derived, Equation 3.20 can be rewritten for pruning a single parameter, w_i, in Equation 3.21.

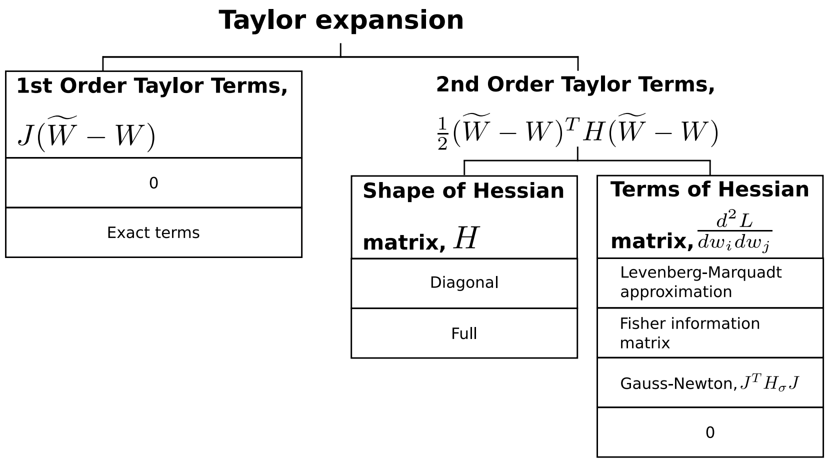
$$\begin{aligned}\mathcal{L}(\widetilde{W}) &\approx \mathcal{L}(W) + \frac{d\mathcal{L}}{dw_i}(0 - w_i) + \frac{1}{2}(0 - w_i)\frac{d^2\mathcal{L}}{dw_i{}^2}(0 - w_i) \\ &\approx \mathcal{L}(W) - w_i\frac{d\mathcal{L}}{dw_i} + \frac{1}{2}w_i{}^2\frac{d^2\mathcal{L}}{dw_i{}^2}\end{aligned} \tag{3.21}$$

The approximation of the sensitivity given by the second order Taylor expansion can be used as a saliency metric (LeCun et al. 1989). Hence, the saliency of a single weight is given by Equation 3.22. Similarly, pruning a set of parameters means setting these parameters to zero in $\widetilde{W}$.

$$S(w_i) = -w_i\frac{\partial\mathcal{L}}{\partial w_i} + \frac{1}{2}w_i{}^2\frac{\partial^2\mathcal{L}}{\partial w_i{}^2} \tag{3.22}$$

Computing Equation 3.20 exactly is very expensive. While the first order term of the equation (the term involving the gradients) is computed in linear time, the higher order terms are more difficult to obtain. The Hessian matrix scales with the quadratic of the number of weights in the network. To better understand the difference in computation and memory cost, let us consider the number of operations to compute each element of a feature map through backpropagation. During backpropagation, a given layer's, l, output feature map gradients are given by the next layer's, $l + 1$, backpropagation. It is the layer's input feature map gradients that are computed during backpropagation of the layer l (see Appendix A for more details). Given a layer l, the gradients $\frac{d\mathcal{L}}{d\,{}^{l}A}$ are known, and $\frac{d\mathcal{L}}{d\,{}^{l-1}A}$ is computed during the l^{th} backward pass. Computing each point of $\frac{d\mathcal{L}}{d\,{}^{l-1}A}$ has a complexity $O({}^{l}m \times ({}^{l}k)^2)$. There are ${}^{l-1}m \times {}^{l-1}height \times {}^{l-1}width$ such points to be computed for the input feature map gradient. To understand the higher complexity of computing the full Hessian matrix, let us consider the computational cost of the layer-wise Hessian using chain rule. If we did a full back propagation of the layer-wise Hessian, we would instead need to compute $\frac{({}^{l-1}m \times {}^{l-1}height \times {}^{l-1}width)^2}{2}$ points each having a complexity $O({}^{l}m \times ({}^{l}k)^4)$.

To reduce computation and storage cost, different approximations can be applied to the different terms of Equation 3.20 (LeCun et al. 1989; Hassibi and Stork 1992; Molchanov et al. 2017; Theis et al. 2018) to obtain different saliency metrics. Popular approximations are presented in Figure 3-5. Table 3.6 summarises various approximations of Equation 3.20 that have been used as saliency metrics.



Taylor expansion
1st Order Taylor Terms, $J(\widetilde{W} - W)$
0
Exact terms
2nd Order Taylor Terms, $\frac{1}{2}(\widetilde{W} - W)^T H(\widetilde{W} - W)$
Shape of Hessian matrix, H
Diagonal
Full
Terms of Hessian matrix, $\frac{d^2L}{dw_i dw_j}$
Levenberg-Marquadt approximation
Fisher information matrix
Gauss-Newton, $J^T H_\sigma J$
0

Figure 3-5: Different approximations can be applied to Equation 3.20 to easily approximate the sensitivity of a set of weights.

Saliency metric	**1st order terms**	**2nd order terms (Hessian)**	
		Shape	Approximation Used
Optimal Brain Damage (LeCun et al. 1989)	Omitted	Diagonal	Levenberg-Marquadt
Optimal Brain Surgeon (Hassibi and Stork 1992)	Omitted	Full	Fisher
First order Taylor (Molchanov et al. 2017)	Exact	Omitted	-
Fisher Information (Theis et al. 2018)	Omitted	Diagonal	Fisher
Collaborative Channel Pruning (Peng et al. 2019)	Omitted	Full	Gauss-Newton with $H_\sigma = diag(L \oslash (P \odot P))$

Table 3.6: Approximations applied to the terms in Equation 3.20 to obtain a saliency metric for pruning.

Even though different approximations are used, the resulting saliency metrics can be very similar. Equation 3.24 is used by Theis et al. (2018) is very similar to Equation 3.23 used by Molchanov et al. (2017).

$$
{}^{l}S_i = \frac{1}{n({}^{l}A_i)} \left| \sum_{a \in {}^{l}A_i} a \frac{d\mathcal{L}}{da} \right| \tag{3.23}
$$

$$
{}^{l}S_i = \frac{1}{2} \left(\sum_{a \in {}^{l}A_i} a \frac{d\mathcal{L}}{da} \right)^2 \tag{3.24}
$$

Consider Only First Order Terms

A first order expansion can also be used to approximate the change in loss. The second order terms in the Equation 3.20 can be set to zero to get the saliency metrics given by considering a first order Taylor expansion. The resulting equation has all its quantities readily available during backpropagation. The Jacobian matrix (i.e.

the matrix of gradients) is computed during backpropagation. Equation 3.23 is used by Molchanov et al. (2017) as a saliency metric.

A first order Taylor approximation is theoretically a coarser approximation of the real function than a second order Taylor expansion. However, it has the advantage of omitting computation of higher order derivatives. In practice, we can see that good saliency metrics can be derived from a first order Taylor expansion (Molchanov et al. 2017; Ding et al. 2019).

Consider Only Second Order Terms

On the other hand, other works choose to neglect the first order term. A common assumption is that if a network has been trained to a local minimum, its first order derivatives will be very close to zero. This assumption about the network's convergence, then allows the first order term (containing the gradients) to be approximated to zero. This a common approximation when using second order Taylor expansions (LeCun et al. 1989; Hassibi and Stork 1992; Theis et al. 2018; Peng et al. 2019; Dong et al. 2017a).

Approximation of Second Order Terms

Computing the estimated terms using a second order Taylor expansion can be expensive due to the cost of computing the Hessian matrix of the loss function for every parameter. To reduce computation cost of the Hessian matrix, the terms that are not on its diagonal can be ignored (Theis et al. 2018; LeCun et al. 1989). Considering only the diagonal terms of the Hessian reduces the number of points to be computed. The number of terms on the diagonal of the Hessian is equal to the number of terms in the Jacobian (gradients). To further reduce the computation cost of the remaining terms, one can use more approximations. The remaining terms of Hessian can be estimated using a Levenberg-Marquardt approximation for each layer of the network (LeCun et al. 1989), the Fisher information (Theis et al. 2018) or a Gauss-Newton approximation. The Levenberg-Marquardt approximation used by Optimal Brain Damage (LeCun et al. 1989) propagate only the diagonal Hessian (see Appendix A). Hence, its computational cost is similar to backpropagating the gradients. Optimal Brain Surgeon (Hassibi and Stork 1992; Hassibi et al. 1993) also use the Fisher matrix as an approximation for the Hessian, however they do not neglect the non-diagonal terms. By including the non-diagonal terms of the Hessian matrix, Optimal Brain Surgeon (Hassibi and Stork 1992) considers the pairwise dependency between parameters.

3.8 Experimental Setup

3.8.1 Saliency metrics

We selected a subset of metrics that can be derived from Table 3.1, excluding $alternateBackprop$ (metrics based on an alternative backpropagation) and $reconstruction$ (metrics based on reconstruction error).

The second order Taylor expansion expressed in Table 3.1 cannot be realistically computed exactly. The approximations seen in Figure 3-5 are used for the second order expansion. We choose to fix the shape of the Hessian to a diagonal matrix and use either an expensive (Levenberg-Marquardt) or a cheap (Gauss-Newton with $H_\sigma = 1$) algorithm to compute the remaining terms. Neglecting the first order terms in a second order Taylor expansion being a popular approximation (LeCun et al. 1989; Hassibi and Stork 1992; Theis et al. 2018; Peng et al. 2019), we test this approximation, leading to a total of 4 different pointwise saliency metrics that use the Hessian.

We test all the saliency metrics that can be derived from the 2 base inputs, 7 different pointwise saliency metrics, 5 different reduction methods and 5 different scaling factors. In total we evaluate 308 different saliency metrics that are the result of combining different components from each of the four parts of our taxonomy. No previous evaluation in the literature has considered so many different saliency metrics. The great majority of the metrics we measure are new combinations of components and have not been evaluated in previous literature.

3.8.2 Pruning Algorithm

Saliency metrics are embedded into a pruning algorithm to remove weights from the network. Pruning algorithms can range from simply removing a fixed number of channels every iteration (Mao et al. 2017), to the use of reinforcement learning (He et al. 2018b), and evolutionary particle filters (Anwar et al. 2017). While state-of-the-art pruning algorithms push the boundaries of pruning further, simple pruning algorithms are still very efficient (Mao et al. 2017; Han et al. 2015; Frankle and Carbin 2019). Simple pruning algorithms heavily rely on how well the saliency metric can predict the least salient entities. Since our goal is to compare saliency metrics to each other with the least number of confounding factors, we opt for simple pruning algorithms. We evaluate the performance of the saliency metrics using the pruning algorithm given in Algorithm 1.

We implement channel pruning so that at every step of the pruning process, we have a dense network with fewer weights than before pruning. Using Algorithm 1, we select a channel to prune from some convolutional layer of the network at each step. Where we would remove a channel which participates in a join-type operation in directed acyclic graph (DAG) structured networks (for example, pruning a channel from one side of a skip connection in ResNet), we also remove the corresponding DAG sibling channels, so that data dependencies are satisfied and the network remains dense. The weights of the dependent channels are included in $\mathcal{TC}({}^{l}W_{i})$.

Algorithm 1 Algorithm for pruning with retraining. Evaluating different channel selections for a CNN with loss function $\mathcal{L}$, accuracy $\mathcal{Y}$, and converged weights W with M channels for a user-defined maximum drop in initial test accuracy, $testAccDrop$, maximum drop in train accuracy $trainAccDrop$ and maximum number of steps to use for retraining, $maxSteps$.

```
$initialTestAcc = \mathcal{Y}(W, \mathcal{I}_{test})$
$initialTrainAcc = \mathcal{Y}(W, \mathcal{I}_{train})$
**repeat**
  Compute ${}^{l}S_i, \forall l \in \{0..l_{max}-1\}, \forall i \in \{0..{}^{l}m-1\}$ using $\mathcal{I}_{val}$
  Get $i$ and $l$, such that ${}^{l}S_i = min({}^{p}S_q), \forall p \in \{0..l_{max}-1\}, \forall q \in \{0..{}^{l}m-1\}$ and
  ${}^{l}W_j$ is a non-zero tensor.
  $W = W - \mathcal{TC}({}^{l}W_j)$
  $retrainingSteps = 0$
  **repeat**
    Retrain with 1 batch of images from $\mathcal{I}_{retrain}$
    $trainAcc = \mathcal{Y}(W, \mathcal{I}_{retrain})$
    $retrainingSteps = retrainingSteps + 1$
  **until** ($trainAcc > initialTrainAcc - trainAccDrop$)
      or ($retrainingSteps \geq maxSteps$)
  $testAcc = \mathcal{Y}(W, \mathcal{I}_{test})$
**until** $testAcc < initialTestAcc - testAccDrop$
```

3.8.3 Datasets

We run our experiments using three different datasets: CIFAR-10 (Krizhevsky 2009), CIFAR-100 (Krizhevsky 2009), and a downsampled ImageNet (Chrabaszcz et al. 2017).

These datasets were chosen as they are commonly used standard image classification datasets with different carateristics. The three datasets used in this chapter all contain 32×32 RGB images. They differ in the number of classes, the number of images per classes, and type of classes. CIFAR-10 contains 50000 train images and 10000 test images classified into 10 different classes of animals and vehicles. CIFAR-100 is a general dataset that contains 50000 train images and 10000 test images classified into 100 different classes. For ImageNet-32, the images from ImageNet (ILSVRC 2012 challenge) (Deng et al. 2009) are downsampled to 32×32 pixels (Chrabaszcz et al. 2017). ImageNet is a general dataset that contains 1.28 million train images and 50000 test images classified into 1000 different classes. ImageNet-32 contains the same number of images and classes as original ImageNet but have each image resized to 32 by 32 pixels.

These three datasets each have their own disjoint training set, $\mathcal{I}_{train}$, and testing set $\mathcal{I}_{test}$. We train the CNNs on the whole training set, $\mathcal{I}_{train}$, and measure their test accuracy using $\mathcal{I}_{test}$.

We split $\mathcal{I}_{train}$ into two disjoint sets $\mathcal{I}_{val}$ and $\mathcal{I}_{retrain}$. $\mathcal{I}_{val}$ is used for computing the saliency metrics for channel pruning. $\mathcal{I}_{retrain}$ is used only during the retraining phase.

3.8.4 CNN models

We conduct a wide range of experiments using different networks and different datasets.

We use the CIFAR-10 dataset on LeNet, CIFAR10 network, ResNet-20, NIN, and AlexNet. We use ResNet-20 (He et al. 2016), and NIN (Lin et al. 2013) as originally described for the CIFAR-10 dataset. We modify the first layer of LeNet-5 (Lecun et al. 1998) and AlexNet (Krizhevsky et al. 2017) to process 32×32 RGB images instead of their original input.

We use the CIFAR-100 dataset on ResNet-20, NIN, and AlexNet. We use ImageNet-32 on AlexNet.

We maintain the same input size for all our networks to 32 by 32 pixels. If a network is used for different datasets, the only structural change we apply to that network is modifying its last layer to classify either 10, 100, or 1000 classes.

These nine networks are trained from scratch and the test accuracy of these networks are given in Table 3.7.

	LeNet-5	**CIFAR10**	**ResNet-20**	**NIN**	**AlexNet**
CIFAR-10	69.4%	72.8%	88.4%	88.3%	84.2%
CIFAR-100	-	-	59.2%	65.7%	54.2%
ImageNet-32	-	-	-	-	39.7%

Table 3.7: Summary of trained network accuracy on CIFAR-10, CIFAR-100, and ImageNet-32.

3.8.5 Hyperparameters

A batch size of 128 is used for measuring the saliency metric and for retraining. It should be noted that we use the same batch size for measuring the saliency metric and for retraining. This is done to easily compare the cost of measuring saliency metrics and retraining. These costs are explained in Section 3.9.5.

The hyperparameters used in this Chapter is given in Table 3.8.

Each pruning experiment is run 8 times. The maximum drop in accuracy, $maxTestAccDrop$, is set to 5%.

Network	N_{val}	$maxSteps$
LeNet	2	256
CIFAR10	2	256
ResNet-20	2	50
NIN	2	25
AlexNet	2	10

Table 3.8: The hyperparameters used for LeNet, CIFAR10, ResNet-20, NIN, and AlexNet with the CIFAR-10, CIFAR-100, and ImageNet-32 datasets.

3.9 Results

We begin by considering the three broad categories of saliency metrics proposed in Figure 3-1. A naturally occurring question is to identify the absolute best-performing method in each of our experimental scenarios, and to determine the best pruning we can obtain of each network on each dataset in experiments.

In Table 3.9, we show the results for the best performing saliency metric in each information category for each network. In this evaluation, we compare the number of weights pruned allowing a drop in top-1 accuracy of at most 5%. When this threshold is exceeded, we stop the experiment and take the last snapshot of the model which was above the threshold as the candidate pruned model. We repeat this experiment for 8 runs, obtain the mean percentage of weights removed and a 95% confidence interval for the mean. This approach is used for all of the experimentation presented.

Network	**Weight-based**	**Feature map-based**	**gradient-based**
CIFAR-10 dataset			
LeNet-5	80.4 ± 2	78.5 ± 2	**84.9** ± 4
CIFAR-10	63.0 ± 12	63.5 ± 4	**67.5** ± 5
ResNet-20	17.6 ± 7	20.6 ± 24	**25.4** ± 9
NIN	63.6 ± 2	59.1 ± 3	**72.8** ± 3
AlexNet	68.0 ± 0.3	69.6 ± 2	**70.1** ± 2
CIFAR-100 dataset			
ResNet-20	5.8 ± 0.4	6.6 ± 1	**12.5** ± 3
NIN	**59.2** ± 1	54.4 ± 2	52.6 ± 1
AlexNet	60.2 ± 0.1	60.2 ± 11	**64.0** ± 3
ImageNet-32 dataset			
AlexNet	**55.4** ± 7	51.6 ± 2	51.7 ± 5

Table 3.9: Effectiveness of metrics which use different information. The maximum sparsity achieved (%) obtained for each information category with Algorithm 1 is shown. The shaded cells correspond to the best results for the given network and dataset.

We can see that gradient-based methods are typically the best-performing in experiments. However, there is no one standout method in our experiments, but rather we see that different saliency metrics obtain the best results on different networks or with different datasets. Table 3.10 shows the saliency metrics corresponding to highlighted results in Table 3.9.

Taking AlexNet as an example, we see that on each of the three classification tasks, a different saliency metric produced the best results. On the CIFAR-10 and CIFAR-100 tasks, a gradient-based metric was ultimately most effective, while on the ImageNet task, a pure weight-based metric won out. The choice of which gradients to consider also had an effect. For CIFAR-10, a weight-oriented metric ($X = W$) was most effective, while on CIFAR-100 a feature map-oriented metric ($X = A$) was most effective.

Network	**Sparsity %**	**Saliency Metric**
CIFAR-10 dataset		
LeNet-5	84.9 ± 4	$\sum_{x\in {}^lA_i} \left(\frac{d\mathcal{L}}{dx}\right)^2$
CIFAR-10	67.5 ± 5	$\sum_{x\in {}^lA_i} \left\lvert -x\frac{d\mathcal{L}}{dx} + \frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{GN}\right\rvert$
ResNet-20	25.4 ± 9	$\frac{1}{n({}^lW_i)}\left(\sum_{x\in{}^lA_i}\frac{d\mathcal{L}}{dx}\right)^2$
NIN	72.8 ± 3	$\frac{1}{n({}^lW_i)}\left(\sum_{x\in{}^lA_i} -x\frac{d\mathcal{L}}{dx}+\frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{GN}\right)^2$
AlexNet	70.1 ± 2	$\frac{1}{n({}^lW_i)}\sum_{x\in{}^lW_i}\left\lvert -x\frac{d\mathcal{L}}{dx}\right\rvert$
CIFAR-100 dataset		
ResNet-20	12.5 ± 3	$\left(\sum_{x\in{}^lA_i}\frac{d\mathcal{L}}{dx}\right)^2$
NIN	59.2 ± 1	$\frac{1}{\Vert{}^lS\Vert_1}\sum_{x\in{}^lW_i}\lvert x\rvert$
AlexNet	64.0 ± 3	$\frac{1}{\Vert{}^lA_i\Vert_0}\sum_{x\in{}^lA_i}\left\lvert\frac{d\mathcal{L}}{dx}\right\rvert$
ImageNet-32 dataset		
AlexNet	55.4 ± 7	$\frac{1}{n({}^lW_i)}\sum_{x\in{}^lW_i}\lvert x\rvert$

Table 3.10: The best performing saliency metric in each scenario of network dataset combination. The sparsity values listed correspond the shaded results in Table 3.9. The saliency metric listed correspond to the saliency metric that achieved this sparsity level.

While these selected results show the best-performing metrics in our experiments, we performed the same evaluation for all 308 candidate saliency metrics. We cannot present data for all 308 experiments, but we summarise the trends from the data in the remainder of this section as *Findings*, which highlight key trends with examples as appropriate. Only a small fraction of these 308 saliency metrics have been explored in previous literature. The data in Tables 3.9 and 3.10 lead us to:

Finding 1 Gradient-based metrics typically perform better than metrics which consider only weights or feature maps.

Metrics which use gradients require a full forward and backward pass of the network to compute those gradients, as opposed to feature map-based methods, which require only a forward pass to compute feature map values, or weight-based metrics which require no computation beyond the application of the saliency metric function to the weight values stored in the model. In our experiments, gradient-based metrics yielded the best pruning or tied for the best pruning in 7 of 9 scenarios with different networks and datasets (Table 3.9).

3.9.1 Weight-based or Feature Map-based Methods (Choice of X)

In Tables 3.11 and 3.12, we present the saliency achieved by each pointwise metric when $X = W$ and $X = A$ respectively. This is an important result because although metrics based on $X = W$ and $X = A$ have been proposed in existing literature, there has been no systematic evaluation of which is better in practice. From Tables 3.11 and 3.12, we see that when using pure weight-based metrics or feature map-based metrics there is not a clear-cut winner. However, when the weight gradients or the feature map gradients, it is almost always preferable to use the feature map gradients.

Finding 2 Pruning using the gradient with respect to the output points more often than not outperforms pruning using the gradient with respect to the weights.

With channel pruning, there is a one-to-one correspondence between groups of weights and groups of output points. However, at finer granularities there is no one-to-one correspondence, but rather a one-to-many correspondence. With this mixing of information at finer granularities, we expect the gradient with respect to output points to be a less reliable signal for non-channel-oriented pruning.

3.9.2 Pointwise Metric (Choice of F)

In order to compare pointwise saliency metrics, we need to fix the reduction R and scaling K to reasonable choices which can be expected to give good results on average. Equation 3.25 shows a common choice in the literature (Mao et al. 2017; Li et al. 2017a; Wang et al. 2018; Han et al. 2015; He et al. 2018b; Guo et al. 2016; Ding et al. 2019). Here the reduction R is the L1-norm, and the scaling factor $K = 1$.

$$^{l}S_i = \sum_{x \in {}^{l}X_i} |f(x)| \tag{3.25}$$

For constructing pointwise metrics, the use of a Taylor expansion around the loss function is very common in the literature. We provide the first experimental evaluation of five Taylor expansions which use different approximations. From the results in Tables 3.11 and 3.12, we observe that some approximations are often poor. In particular, neglecting the first order terms (the gradient) when using a second order Taylor expansion around the loss function is a poor approximation in most cases. The degree to which the training process has converged before pruning affects the magnitudes of gradients, meaning they may still be quite large in many cases, so assuming that they are universally close to zero can be a very coarse approximation.

Finding 3 First order terms are often not negligible in second order Taylor expansions.

Metric	w	$\frac{d\mathcal{L}}{dw}$	$-w\frac{d\mathcal{L}}{dw}$	$-w\frac{d\mathcal{L}}{dw}+\frac{w^2}{2}\frac{d^2\mathcal{L}}{dw^2}_{GN}$	$\frac{w^2}{2}\frac{d^2\mathcal{L}}{dw^2}_{GN}$	$-w\frac{d\mathcal{L}}{dw}+\frac{w^2}{2}\frac{d^2\mathcal{L}}{dw^2}_{LM}$	$\frac{w^2}{2}\frac{d^2\mathcal{L}}{dw^2}_{LM}$
Network	**Weights only**	**Weights, input images and labels**					
		Gradients only	**Taylor expansions**				
			1st order	**2nd order with diagonal Hessian**			
				Gauss-Newton approximation		**Levenberg-Marquardt approximation**	
CIFAR-10 dataset							
LeNet-5	78.7 ± 9.8	64.3 ± 13.9	**80.6** ± 3.0	80.3 ± 3.8	79.0 ± 8.0	**80.8** ± 2.8	78.5 ± 5.2
CIFAR10	16.7 ± 62.9	19.3 ± 19.7	53.0 ± 24.5	50.7 ± 24.2	**61.7** ± 17.6	50.4 ± 28.7	22.3 ± 5.6
ResNet-20	1.7 ± 0.0	**6.7** ± 8.7	4.6 ± 3.7	4.6 ± 2.8	4.0 ± 2.3	4.4 ± 4.2	1.2 ± 0.1
NIN	34.5 ± 0.5	5.6 ± 1.8	**61.1** ± 2.8	20.1 ± 59.1	13.7 ± 71.5	60.3 ± 2.3	42.8 ± 1.7
AlexNet	**64.0** ± 9.6	40.1 ± 4.8	51.7 ± 9.9	52.0 ± 9.8	49.7 ± 8.6	55.1 ± 6.1	20.0 ± 24.8
CIFAR-100 dataset							
ResNet-20	3.4 ± 0.2	3.2 ± 3.8	2.8 ± 6.1	**4.1** ± 5.0	1.9 ± 14.0	**4.1** ± 3.4	1.1 ± 0.1
NIN	42.2 ± 1.0	35.2 ± 0.2	36.2 ± 0.1	36.4 ± 1.0	37.1 ± 0.1	36.2 ± 0.9	**52.4** ± 2.7
AlexNet	**58.6** ± 21.4	26.4 ± 14.5	56.1 ± 14.8	52.0 ± 19.3	50.2 ± 11.2	52.1 ± 5.7	27.0 ± 7.3
ImageNet-32 dataset							
AlexNet	28.2 ± 2.1	31.0 ± 3.3	31.1 ± 0.8	31.2 ± 2.2	**35.2** ± 1.0	31.2 ± 0.8	1.1 ± 0.1

Table 3.11: Comparison between different pointwise saliency metrics. The maximum proportion of weights removed (sparsity) by Algorithm 1 (%) using different pointwise saliency metrics with weights as the input ($x = w$) and Equation 3.25 as reduction method. The shaded results correspond to the best pruning results obtained per scenario with $x = w$ and Equation 3.25 as reduction method.

Metric	a	$\frac{d\mathcal{L}}{da}$	$-a\frac{d\mathcal{L}}{da}$	$-a\frac{d\mathcal{L}}{da} + \frac{a^2}{2}\frac{d^2\mathcal{L}}{da^2}_{GN}$	$\frac{a^2}{2}\frac{d^2\mathcal{L}}{da^2}_{GN}$	$-a\frac{d\mathcal{L}}{da} + \frac{a^2}{2}\frac{d^2\mathcal{L}}{da^2}_{LM}$	$\frac{a^2}{2}\frac{d^2\mathcal{L}}{da^2}_{LM}$
Network	**Weights and input images**	**Weights, input images and labels**					
		Gradients only	**Taylor expansions**				
			1st order	**2nd order with diagonal Hessian**			
				Gauss-Newton approximation		**Levenberg-Marquardt approximation**	
CIFAR-10 dataset							
LeNet-5	77.1 ± 2.5	82.0 ± 2.1	82.5 ± 2.2	**82.6** ± 2.7	74.6 ± 2.3	82.1 ± 1.8	69.7 ± 5.6
CIFAR10	56.0 ± 14.5	56.7 ± 19.6	65.4 ± 15.0	**66.8** ± 16.0	57.3 ± 21.6	65.3 ± 15.9	22.3 ± 17.2
ResNet-20	5.5 ± 3.2	4.2 ± 3.5	11.5 ± 14.6	12.8 ± 17.3	9.5 ± 14.1	**13.2** ± 13.4	2.4 ± 0.3
NIN	32.9 ± 25.4	5.4 ± 61.8	58.3 ± 4.3	56.9 ± 4.1	**62.7** ± 14.2	59.7 ± 12.6	38.7 ± 24.6
AlexNet	**69.2** ± 4.4	63.9 ± 5.6	65.0 ± 3.0	63.3 ± 2.4	57.0 ± 5.4	63.8 ± 4.3	35.3 ± 5.1
CIFAR-100 dataset							
ResNet-20	3.8 ± 1.7	2.4 ± 1.0	4.6 ± 2.4	4.7 ± 2.8	**5.2** ± 2.3	4.6 ± 1.7	6.4 ± 4.3
NIN	42.6 ± 2.2	35.2 ± 0.0	36.2 ± 0.1	36.2 ± 0.0	37.0 ± 0.8	36.1 ± 0.1	**45.6** ± 2.0
AlexNet	57.6 ± 6.7	**62.9** ± 2.2	62.7 ± 3.3	**62.9** ± 2.7	52.5 ± 5.8	62.4 ± 3.7	33.1 ± 23.6
ImageNet-32 dataset							
AlexNet	19.4 ± 4.1	**40.4** ± 1.9	30.3 ± 1.2	30.2 ± 1.5	25.6 ± 0.9	30.4 ± 1.7	2.9 ± 1.4

Table 3.12: Comparison between different pointwise saliency metrics. The maximum proportion of weights removed (sparsity) by Algorithm 1 (%) using different pointwise saliency metrics with output points as the input ($x = a$) and Equation 3.25 as reduction method. The shaded results correspond to the best pruning results obtained per scenario with $x = a$ and Equation 3.25 as reduction method.

The second order term in the Taylor expansion $-x\frac{d\mathcal{L}}{dx} + \frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}$ corresponds to the Hessian of the loss function, which is very expensive to compute. Several approximations of the Hessian have been used in the literature. Using a Levenberg-Marquardt approximation requires a very expensive backward propagation of the second order derivatives. This is not commonly implemented in popular deep learning frameworks, because only the first derivative (i.e. the gradient) is required for training. We found that, in the majority of cases in our experiments, a Gauss-Newton approximation of the Hessian is sufficient for pruning. In Tables 3.11 and 3.12, we see that the Levenberg-Marquadt approximation is only clearly advantageous in one case, when pruning NIN on CIFAR-100.

Finding 4 The Gauss-Newton approximation of the Hessian is sufficiently accurate for pruning.

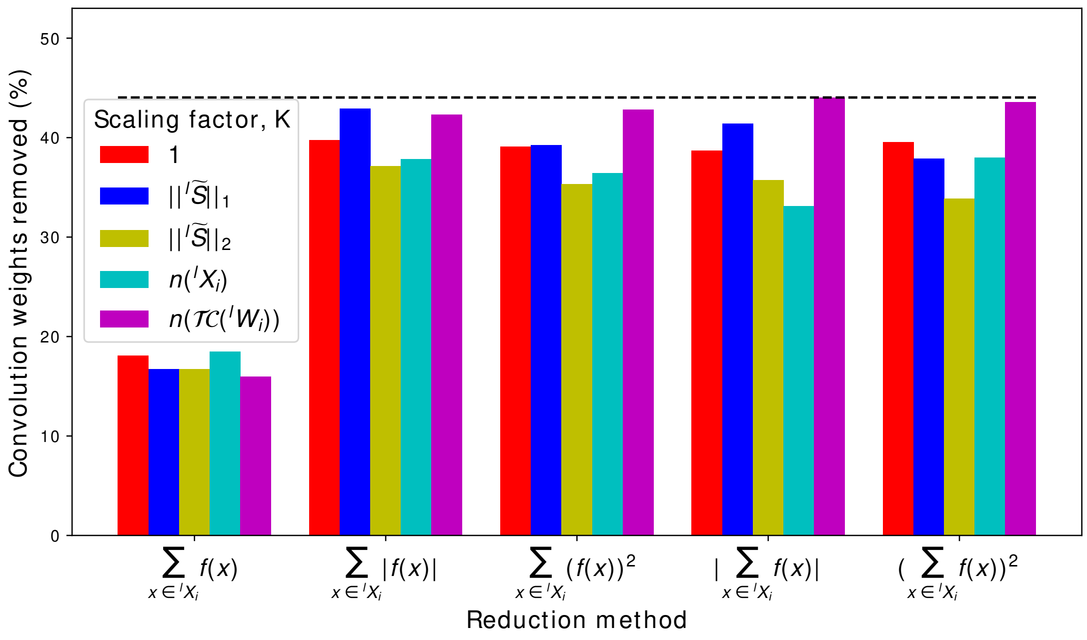
3.9.3 Reduction (Choice of R)

When pruning blocks of weights, the pointwise metrics are combined into a single value using some reduction function. Existing research on pruning often places little focus on reduction and scaling methods, and it can sometimes be difficult to identify the approach used in any given method. Nonetheless, the method by which the pointwise metric is reduced and scaled can greatly influence the quality of a saliency metric.

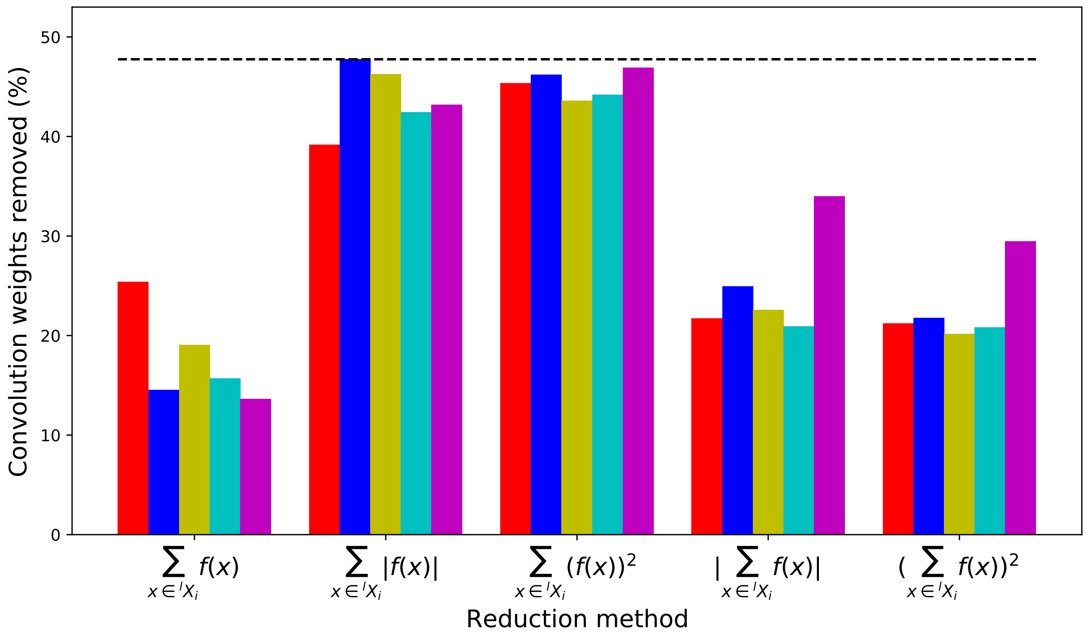
Finding 5 Strictly positive saliency metrics offer better pruning results.

Figure 3-6 presents the results of our experimentation grouped so that only the reduction and scaling methods vary. Figure 3-6a presents a summary for all choices of input and all pointwise metrics. We then examine the three families of metrics outlined in Figure 3-1 in more detail: metrics which consider static information (i.e. weights) only ($X = W$, Figure 3-6b), metrics which consider information available with only a forward pass, i.e. weights and output feature maps, ($X = A$, Figure 3-6c) and finally metrics which consider all information available from a forward and backward pass ($X = A$ and $f(x)$ involves a gradient, Figure 3-6d).

As a general trend, we see that using the same pointwise saliency metric but varying the reduction or scaling methods can produce very different results. For



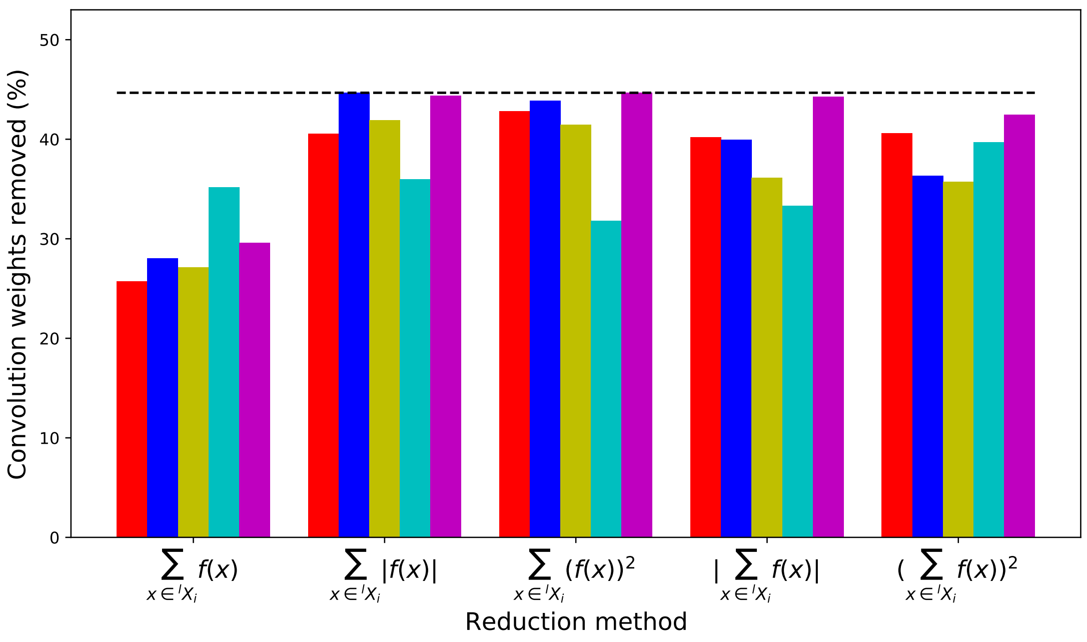
(a) For all tested X and $f(x)$.



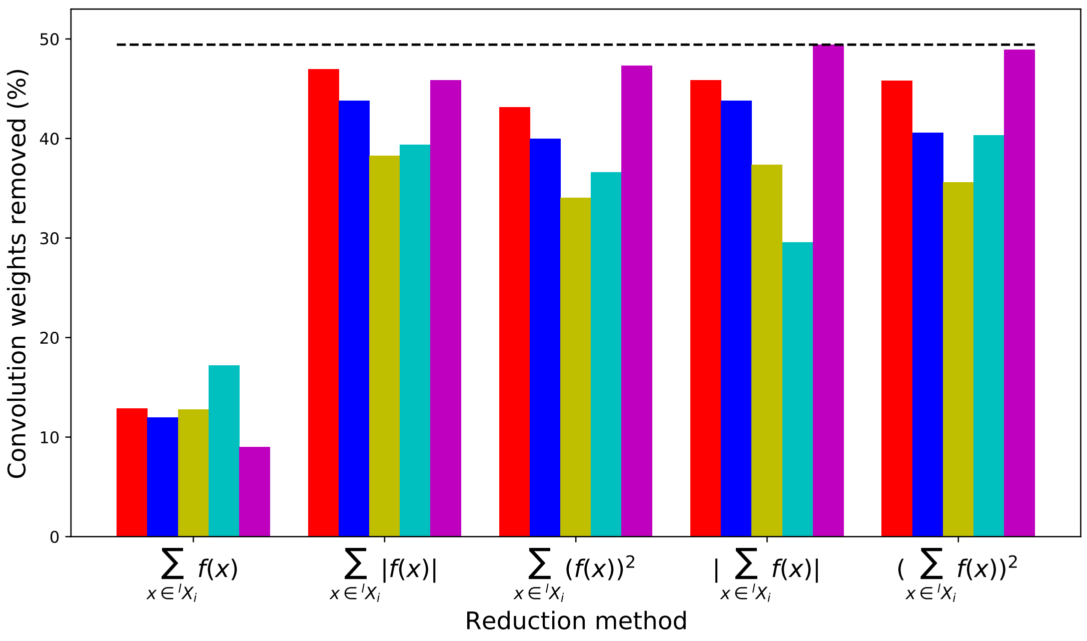
(b) $X = W$ and $f(x) = x$.

Figure 3-6: Comparison between different reduction methods and scaling factors. The percentage of convolution weights removed (sparsity) on average for different scaling methods across different networks and datasets.

example, we see that using the raw sum of the gradients (first bar set in each graph) typically results in poorly performing metrics versus other reduction methods. In



(c) $X = A$ and $f(x) = x$



(d) $X = A$ and $f(x) = -x\frac{d\mathcal{L}}{dx} + \frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{GN}$

Figure 3-6: Comparison between different reduction methods and scaling factors. The percentage of convolution weights removed (sparsity) on average for different scaling methods across different networks and datasets.

each scenario, reducing by the sum of the absolute values of the gradients produces significantly better results (second bar set in each graph).

We observe that the gap between the simple summation and the other guaranteed-positive reduction methods is smaller in Figure 3-6c where we use $X = A$, i.e. the gradient with respect to output features. In our experiments, networks containing ReLU layers are optimised such that ReLU is fused with the convolution layer, meaning the resulting output features are non-negative. This means the simple summation is also guaranteed non-negative, which improves the quality of pruning decisions significantly.

3.9.4 Scaling (Choice of K)

In Figures 3-6b and 3-6c, we see that two scaling methods stand out when we ignore the typically poorly-performing first bar group which corresponds to the simple-summation reduction method. Across all four remaining guaranteed-positive reduction methods, two scaling factors trade blows for first and second place in terms of the quality of pruning decisions: $\left\|{}^{l}\widetilde{S}\right\|_1$ and $n(\mathcal{TC}({}^{l}W_i))$.

Recall from Section 3.3.4 that $\left\|{}^{l}\widetilde{S}\right\|_1$ is the layer-wise L1 norm of saliency values, and $\mathcal{TC}({}^{l}W_i)$ denotes the entire set of weights *transitively* removed when ${}^{l}W_i$ is removed from the network. Both of these scaling factors incorporate structural information, as opposed to strictly local information about the parameters being pruned.

Finding 6 Incorporating structural information of the network in the scaling factor offers better pruning results.

In Figure 3-6b, we can directly compare scaling by the local ($n({}^{l}W_i)$) and transitive ($n(\mathcal{TC}({}^{l}W_i))$) number of weights removed. These are the second-to-last and last bars in each bar set, respectively. While the improvement of using $n(\mathcal{TC}({}^{l}W_i))$ is sometimes small, it is strictly better than using only the local information in each case when considering guaranteed-positive reduction methods.

In Figure 3-6d, we look at how a good gradient-based pointwise metric can be affected by reduction and scaling. Similarly to non-gradient-based metrics in Figures 3-6b and 3-6c, we see that the use of a reduction method that is guaranteed positive led to significantly better results. The gradient-based metric results further highlight the benefit of using non-local information. The best overall pruning result is achieved using $n(\mathcal{TC}({}^lW_i))$ as the scaling factor (fourth bar set in the figure), but using the local scaling factor $n({}^lW_i)$ while keeping everything else fixed makes the quality of the metric plummet – in this case, by nearly 20 percentage points average sparsity achieved across all networks and datasets in our experiments.

Finding 7 The number of weights transitively removed is a better scaling factor than the number of weights locally removed.

3.9.5 Saliency Metrics and Retraining Iterations

Retraining is a crucial step in many pruning algorithms because it allows the network to adjust remaining parameters to compensate for the damage done by the removal of pruned parameters. However, when evaluating saliency metrics, retraining is a confounding factor because it can arbitrarily change weight values. In fact, the effect of retraining is so great that we can often compensate for the suboptimal choices made by poor saliency metrics with enough retraining.

Confounding factors notwithstanding, it seems intuitive that a better saliency metric, should greatly reduce the effort spent on retraining to achieve a given target accuracy and sparsity. Better saliency metrics do less damage to the network to attain a given minimum sparsity, and conversely also result in higher achievable sparsity ratios for a given minimum accuracy. Thus, we expect the total computational cost of pruning (retraining included) to be reduced by choosing a higher-quality saliency metric.

Algorithm 2 Algorithm for pruning without retraining. Evaluating different channel selections for a CNN with loss function $\mathcal{L}$, accuracy $\mathcal{Y}$, and converged weights W with M channels for a user-defined maximum drop in initial test accuracy, $maxTestAccDrop$.

```
$initialTestAcc = \mathcal{Y}(W, \mathcal{I}_{test})$
repeat
    Compute ${}^lS_i, \forall l \in \{0..l_{max}-1\}, \forall i \in \{0..{}^lm-1\}$ using $\mathcal{I}_{val}$
    Get $i$ and $l$, such that ${}^lS_i = min({}^pS_q), \forall p \in \{0..l_{max}-1\}, \forall q \in \{0..{}^lm-1\}$ and
    ${}^lW_j$ is a non-zero tensor.
    $W = W - \mathcal{TC}({}^lW_j)$
    $testAcc = \mathcal{Y}(W, \mathcal{I}_{test})$
until $testAcc < initialTestAcc - testAccDrop$
```

We examine the relationship between the quality of a saliency metric and the total computational cost of pruning. As a proxy for the quality of a saliency metric we use the maximum sparsity achieved using that metric *without* any retraining, i.e. using Algorithm 2. The pruning algorithm outlined in Algorithm 2 is similar to our previous pruning algorithm, except for the omission of retraining steps.

We refer to the total cost of pruning as the total computational cost to reach a certain sparsity while maintaining a fixed accuracy target. Hence, in addition to the accuracy threshold in Algorithm 1, we add a stopping condition related to the sparsity. When this target is met, we stop the experiment and record the total number of steps that were required to prune and retrain the network. The target sparsity ratio chosen for each network on each dataset was the best achievable sparsity from Table 3.9 minus 5%. This additional target allows us to filter out very poor saliency metrics and to compare pruned networks of similar size and accuracy.

The total cost of pruning is given by the sum of the cost of computing the saliency metric and the cost of retraining. The cost of a single retraining step is the cost of one backward and one forward pass of the network. To easily compare cost, we assume that the cost of a backward pass is twice that of a forward pass. The cost of computing a saliency metric is given in Table 3.13. The cost of backpropagating the second order derivatives is similar to the cost of propagating the gradients (LeCun et al. 1989) (see Appendix A).

Pointwise Metric		**Cost**	**Pointwise Metric**	**Cost**
x	$x = w$	0	$-x\frac{d\mathcal{L}}{dx} + \frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{GN}$	$3 \times N_{val}$
	$x = a$	$1 \times N_{val}$	$\frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{GN}$	$3 \times N_{val}$
$\frac{d\mathcal{L}}{dx}$		$3 \times N_{val}$	$-x\frac{d\mathcal{L}}{dx} + \frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{LM}$	$5 \times N_{val}$
$-x\frac{d\mathcal{L}}{dx}$		$3 \times N_{val}$	$\frac{x^2}{2}\frac{d^2\mathcal{L}}{dx^2}_{LM}$	$5 \times N_{val}$

Table 3.13: Cost of saliency metrics using $\mathcal{I}_{val}$ (N_{val} batches of images).

In Figure 3-7, we see the result of this experiment for AlexNet on the CIFAR-10 dataset. Each point on the graph is one saliency metric. We see that many saliency metrics are able to be used in Algorithm 2 to meet both the sparsity and accuracy targets. However, as predicted, poorer saliency metrics result in much more retraining being required to reach a network of the given quality than good saliency metrics. To quantify our results in terms of the correlation of metric quality and pruning cost, we use the Spearman rank correlation. The correlation for AlexNet on CIFAR-10 (as presented in Figure 3-7 is -0.9. A similar trend is observed (negative correlation in Table 3.14) for the other networks except for ResNet-20 on CIFAR-10 where we had only 3 metrics meeting the targets. This leads us to another novel finding of our work.

CIFAR-10				
LeNet-5	**CIFAR10**	**ResNet-20**	**NIN**	**AlexNet**
-0.10 ($3e^{-1}$)	-0.5 ($8e^{-4}$)	1 (0)	-0.7 ($6e^{-2}$)	-0.9 ($2e^{-19}$)
CIFAR-100				
-	-	-0.6 ($1e^{-1}$)	-0.6 ($4e^{-1}$)	-0.8 ($4e^{-18}$)
ImageNet-32				
-	-	-	-	-0.9 ($3e^{-3}$)

Table 3.14: Spearman rank correlation (with p-value) between metric quality and computational cost of pruning including retraining.

Finding 8 Better saliency metrics greatly reduce retraining requirements in pruning.

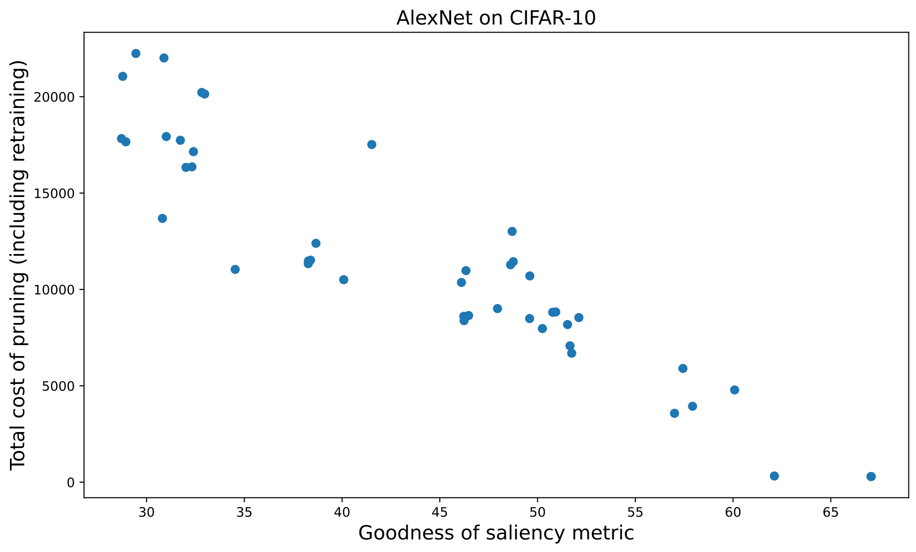


Figure 3-7: Cost to prune a network (lower is better) against quality of the saliency metric (higher is better).

Using a good saliency metric can greatly reduce the total cost for pruning a network. In the case of AlexNet trained on CIFAR-10 Figure 3-8 , with a fixed sparsity and accuracy target and starting from the same initial trained network, the most cost efficient experiment was about 75× less computationally expensive than the experiment with the highest computational cost. The smallest improvement is observed for NIN on CIFAR-10 with the most best metric being 1.4× less computationally expensive that the worst case.

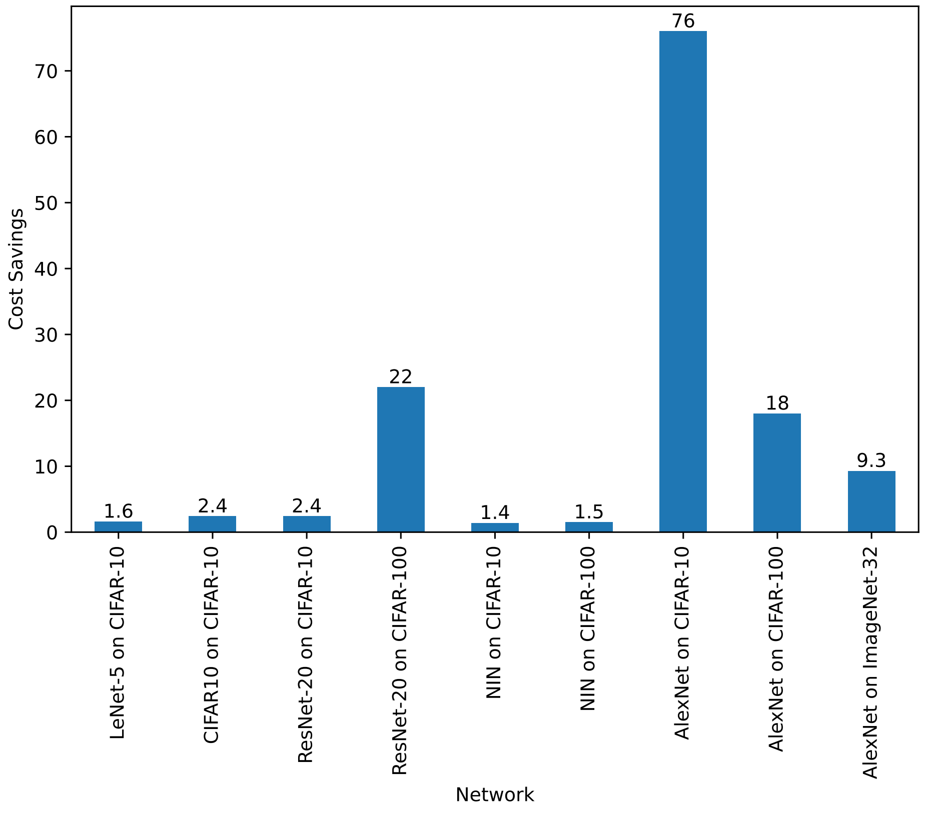


Figure 3-8: Relative factor (× times) cost for pruning between the most expensive and least expensive pruning experiment that meet the sparsity and accuracy targets.

We observe that for some networks the least computationally expensive pruning experiment needed only a few retraining steps to meet the accuracy and sparsity requirements. In Figure 3-9 , we see that for AlexNet on CIFAR-10 and on ImageNet-32 as well as ResNet-20 on CIFAR-20, the most cost efficient cases require almost no retraining steps to meet the sparsity and accuracy targets.

With good saliency metrics, pruning algorithms can reduce their computational cost while meeting their accuracy and sparsity targets.

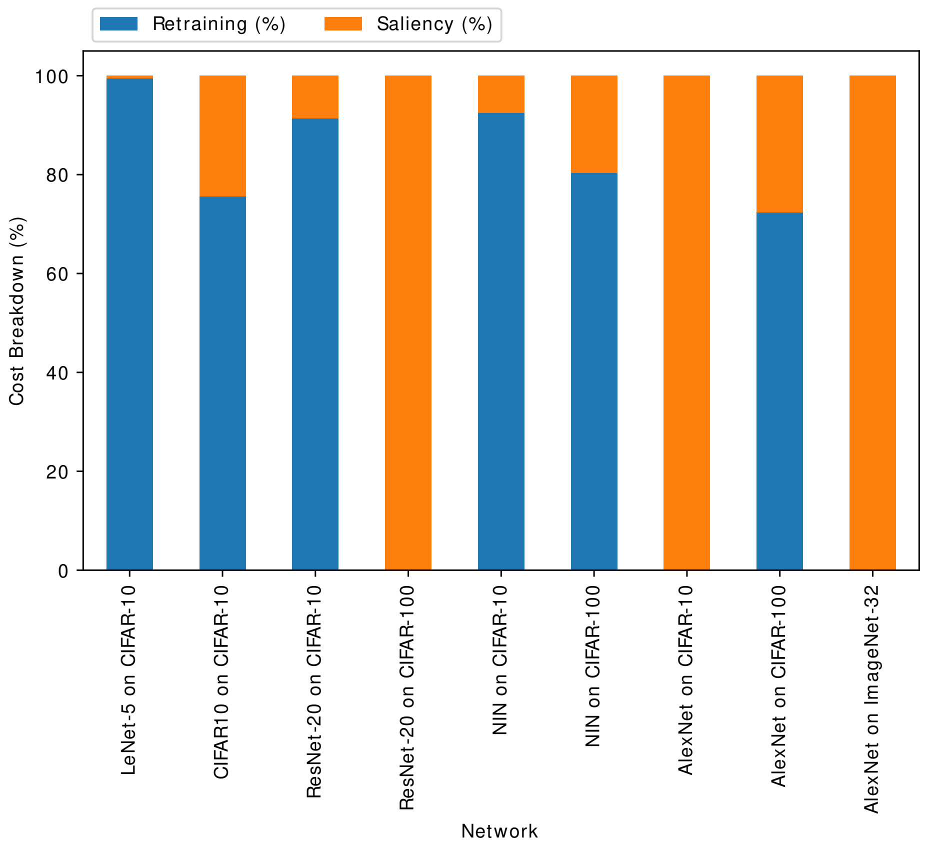


Figure 3-9: Proportion of computational cost for saliency computation and retraining for the case with smallest total cost that meets the accuracy and sparsity requirements.

3.10 Conclusion

Although many pruning strategies have been proposed, we have identified that a common element is the saliency metric that seeks to identify unimportant parameters. We propose a novel taxonomy that characterises saliency metrics, by combining elements from four independent components: (1) base input, (2) pointwise metric, (3) reduction, and (4) scaling. This taxonomy allows us to identify common components among saliency metrics in the many existing pruning strategies, and to derive novel metrics by combining elements from four independent components. We experimentally evaluate 308 such metrics.

We confirm some well-known results, like that gradient-based methods are significantly better than simpler methods based purely on the weights or output feature maps. But we also find new insights that were previously unknown despite a large existing literature on pruning. For example, metrics that use the gradient with respect to outputs tend to outperform those using the gradient with respect to weights. The most successful gradient-based methods use a second-order Taylor expansion. Within this expansion, the first order term contains important information that should not be omitted as is the practice in metrics such as Optimal Brain Damage (LeCun et al. 1989), but the Gauss-Newton approximation is sufficient for the second order term. Another important new insight is that good metrics can be easily undermined by a poor reduction, such as simply adding pointwise terms. On the other hand, there is scope for significant improvements in scaling factors containing structural information, such as our novel scaling method based on the number of transitively-pruned parameters that arise when pruning channels.

The saliency metric is just one component of pruning algorithms but it has a critical impact on the success of pruning. We anticipate that our taxonomy and evaluation will guide practitioners to the best existing saliency metrics, and direct researchers to open new frontiers in the design space. In particular, methods of reduction and scaling are perhaps the least developed aspect of saliency metrics, and are likely offer the greatest opportunities for improvement.

Chapter 4

Composition of Saliency Metrics

Traditionally, a single heuristic saliency metric is used for the entire pruning process. In this chapter, we show how to compose a set of these saliency metrics to form a much more robust (albeit still heuristic) saliency. The key idea proposed in this chapter is to exploit the cases where the different base metrics do well, and avoid the cases where they do poorly by switching to a different metric. Since the values of saliency metrics cannot be compared directly, we use a simple oracle to combine the decisions of the saliency metrics. Using the same experimental setup as Section 3.9.5 on the CIFAR-10 and CIFAR-100 datasets, we show that the composite saliency metrics derived by our method consistently match or outperform all the individual constituent metrics.

4.1 Introduction

A wide variety of heuristic saliency metrics have been proposed over decades of research in artificial intelligence(Han et al. 2015; LeCun et al. 1989; Hassibi and Stork 1992; Mozer and Smolensky 1988; Molchanov et al. 2017). As seen in Chapter 3, each of these heuristic saliency metrics may perform better or worse in context, with no one metric being clearly superior. When pruning a neural network, one must often resort to simple rules of thumb or guesswork to select an appropriate heuristic to guide the pruning process. The use of a single saliency metric for the entire pruning process also assumes that a single saliency metric accurately approximates the importance of weights for the entire pruning process. However, the underlying assumptions of the chosen saliency metric may be invalidated at some point of the pruning process and another saliency metric may be able to estimate the importance of the weights more accurately.

4.1.1 Contributions

In this chapter, we propose a method to derive a *composite saliency metric* which can avoid poor choices made by otherwise effective *constituent* saliency metrics. Our approach uses a *myopic oracle* to decide which of a fixed set of constituent metrics should be active at every step of the pruning process. As the predictive power of the constituent metrics waxes and wanes, our approach dynamically switches between metrics so that the most appropriate metric is guiding the process at all points.

We make the following principal contributions:

- We propose a composite saliency metric that uses existing saliency metrics.
- We use the constituent saliency metrics to narrow the search space of pruning.
- We show how to compose different saliency metrics automatically using the myopic oracle.
- We experimentally investigate fusion of state-of-the-art saliency metrics using the CIFAR-10 and CIFAR-100 datasets on 5 different network architectures.

4.2 Background

Most saliency metrics rely on some assumptions. For example, when using the L1-norm of weights, the assumption is that smaller weights contribute less to the network. Their underlying assumptions can sometimes be conflicting. The 2^{nd} order Taylor expansion using Fisher information and 1^{st} order Taylor expansion are both derived using the Taylor expansion presented in Figure 4-1. However, they are constructed under different assumptions. The construction of the 2^{nd} order Taylor expansion using Fisher information (Theis et al. 2018) assumes that the weight gradients and feature map gradients are insignificant. Hence, the 1^{st} order terms in Figure 4-1 are ignored and the second order terms are approximated to the Fisher information to derive Equation 4.6. On the other hand, when using a first order Taylor expansion, the higher order terms are considered insignificant, meaning we ignore the 2^{nd} order terms in Figure 4-1 to derive Equation 4.5.

$$\mathcal{L}(A - {}^{l}A_i) - \mathcal{L}(A) \approx \underbrace{\boxed{\frac{d\mathcal{L}}{dA}(-{}^{l}A_i)}}_{\text{1st order terms}} + \underbrace{\boxed{\frac{1}{2}(-{}^{l}A_i)^T \frac{d^2\mathcal{L}}{dA^2}(-{}^{l}A_i)}}_{\text{2nd order terms}}$$

Figure 4-1: Estimating the effect of pruning the i^{th} output channel of the l^{th} layer of a network with loss function $\mathcal{L}$ using a 2nd order Taylor development around A, the feature maps of the network.

This is a crucial distinction, because these built-in assumptions in the construction of the metrics are typically not simultaneously true for any given network. Moreover, as the pruning process continues, the degree of significance of different components can, and does, change. When most of the remaining information is in higher order components, metrics using only first order components are effectively making random decisions, and vice versa.

For example, in the case of pruning a partially converged network, the gradients of the weights and feature maps are very unlikely to be negligible. When pruning a fully converged network, the gradients are much more likely to be negligible. When pruning a fully trained network, we start with converged weights. However, as the pruning process proceeds, we may end up pruning partially converged weights, since the pruning process degrades the network.

4.3 Composing Saliency Metrics

When using any one saliency metric for the entire pruning process, we run the risk of the metric assumptions being invalidated, leading to poor decisions being made by the metric. Ideally we could combine the best aspects of different saliency metrics. The chief difficulty lies in combining the numerical output of different pruning metrics, which are not directly comparable.

Consider the application of two saliency metrics A and B to neural network $\mathcal{N}$ with a total of four weights $[a, b, c, d]$. Let us suppose that ranking the weights with each metric yields the rankings $A(\mathcal{N}) = [0.5, 0.4, 0.9, 0.1]$ and $B(\mathcal{N}) =$

$[0.01, 0.05, 0.04, 0.06]$. Metric A indicates removing weight d will have the least effect, while metric B indicates removing weight a will have least effect.

If the metrics agree on the weights to be removed, there is no issue. However, if they disagree, there are two alternatives to consider. If either choice results in the same amount of damage to the network, we can call the disagreement trivial. However, if one causes more damage than the other, the disagreement is non-trivial. If we continue to use the suboptimal metric to guide the process, we will introduce more and more relative error.

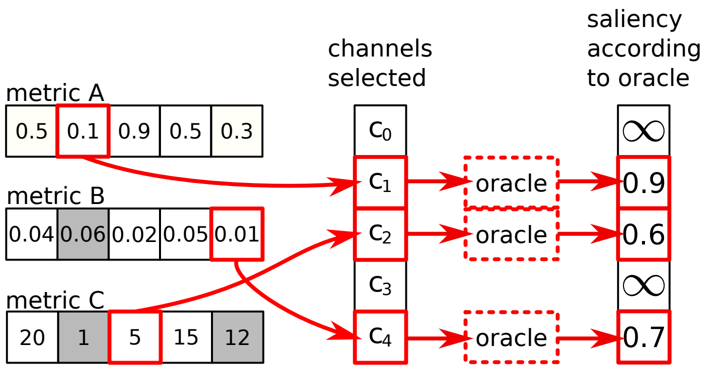
Inspection of the numerical saliency values assigned to each weight or set of weights by metrics A and B exposes the difficulty of combining these metrics numerically. Although each metric ranks the weights or set of weights in the network, in principle, these saliency values can have arbitrary scales. If we were to combine the metrics with a simple linear combination, such as a weighted average, one metric would be disproportionately selected. As we add more and more metrics to the set, the difficulty increases. However, by performing a forward pass of the network, we may determine at any point the true effect on the loss function of making a particular pruning decision. This is the key to our proposed approach.

4.3.1 Our Proposed Method: Myopic Oracle

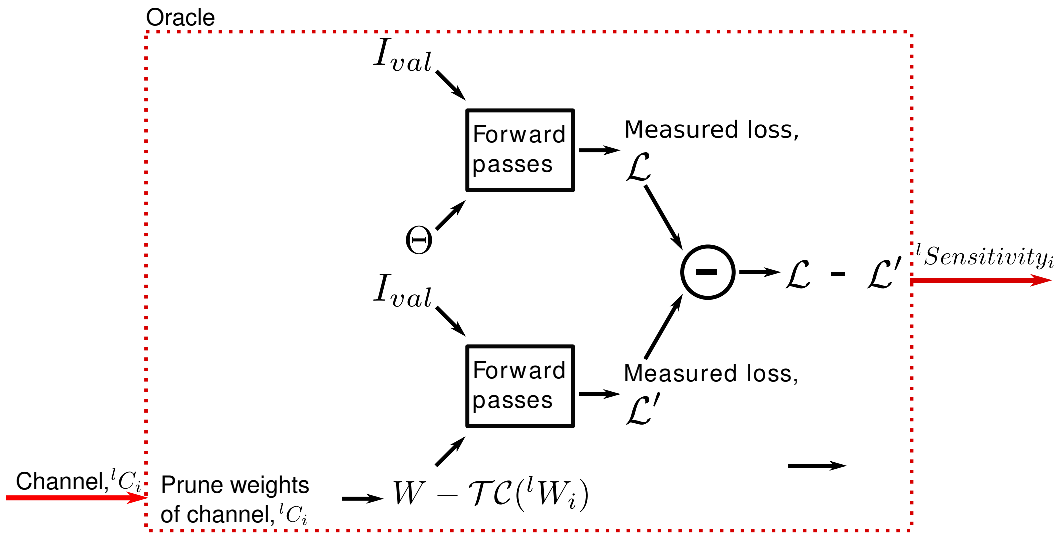
When different saliency metrics yield different rankings for the same sets of weights, we can evaluate which ranking is the most correct by performing a direct measurement of the *sensitivity* of the network to the removal of the proposed subsets of parameters.

For brevity of presentation, in the following treatment we choose subsets of parameters corresponding to whole output feature maps (i.e. channel pruning). However, the approach is no different for other subsets of weights (filters, individual weights or any other granularity of pruning).

For a CNN characterised by the loss function $\mathcal{L}$ and permanent weights W, the sensitivity of the i^{th} channel of the l^{th} layer the network, using forward passes on the validation set I_{val}, is given by the change in the loss caused by pruning that channel, as shown in Equation 4.1.



(a) Channel selection ($k = 3$) and oracle evaluation. This figure also illustrates how the myopic oracle selects the best channels out of the remaining ones in a round-robin manner. The shaded channels for metric B and C represent channels previously chosen for evaluation by the oracle.



Oracle
I_{val}
Forward passes
Measured loss, $\mathcal{L}$
Θ
I_{val}
Forward passes
Measured loss, $\mathcal{L}'$
$\mathcal{L}$ - $\mathcal{L}'$
${}^{l}Sensitivity_i$
Channel, ${}^{l}C_i$
Prune weights of channel, ${}^{l}C_i$
$W - \mathcal{TC}({}^{l}W_i)$

(b) Sensitivity evaluation by the oracle for a channel, c, according to Equation 4.1.

Figure 4-2: Combining rankings of saliency metrics A, B, and C using a myopic oracle with $k = 3$.

We remind some relevant notation to facilitate the description of the sensitivity measure used. Figure 4-3 provides a summary of the notation used. $\mathcal{TC}({}^{l}W_i)$ denotes all the parameters that need to be removed when the c^{th} channel of the l^{th} layer is removed. Removing $\mathcal{TC}({}^{l}W_i)$ from the network results in a dense network with fewer weights. Hence, $\mathcal{TC}({}^{l}W_i)$ contains ${}^{l}W_i$ but also includes parameters from other layers that interact with the selected channel. The case of a simple forward feed network is shown in Figure 4-3, where $\mathcal{TC}({}^{l}W_i)$ is highlighted in red or green, and ${}^{l}W_c$ is highlighted in red. An in-depth explanation of how we obtain $\mathcal{TC}({}^{l}W_i)$ is given in Section 5.3.1.

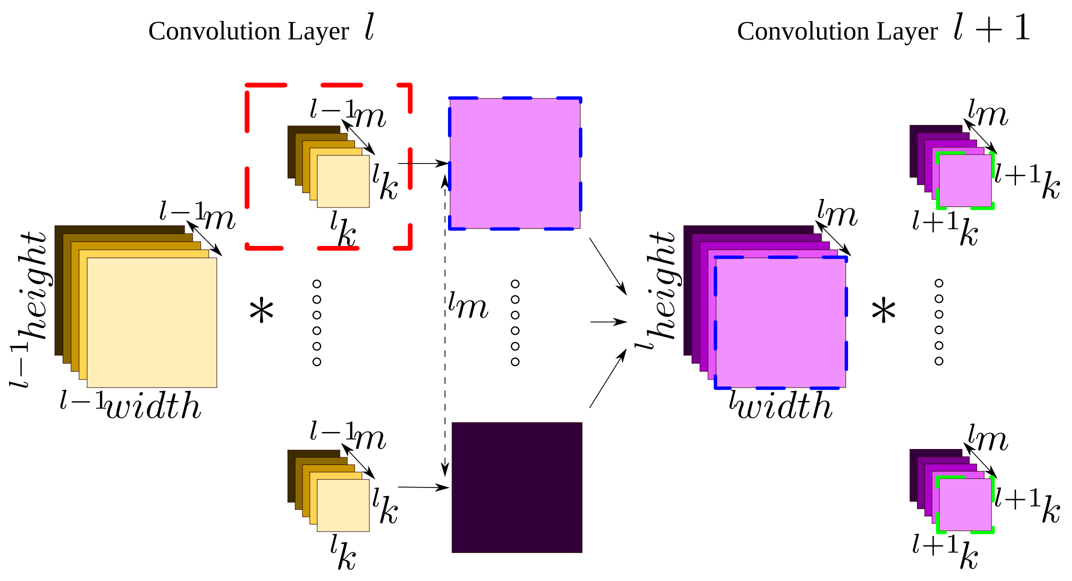


Figure 4-3: Reminder of Figure 3-3. The complete description of our notation is described in Section 2.2.1.

$$^{l}Sensitivity_i = \mathcal{L}(W - \mathcal{TC}(^{l}W_i), I_{val}) - \mathcal{L}(W, I_{val}) \tag{4.1}$$

At every pruning step, the *myopic oracle* measures the sensitivity of only k different channels using the validation set. Notionally, k is the number of channels that the myopic oracle can "see". The choice of the value of k depends on the pruning algorithm used, but must be at least the number of channels that the pruning algorithm considers pruning simultaneously. Hence, k can vary depending on the pruning algorithm.

It should be noted that the different constituent saliency metrics and the sensitivity computed by the myopic oracle use the same dataset, I_{val}, containing N_{val} batches of images. The cost of running the myopic oracle for one channel is similar to the cost of computing the feature map-based heuristics that use forward passes only. Hence, if N_{val} batches are used to measure the sensitivity for each channel the cost of running the oracle (excluding the cost of computing the individual saliency metrics) is $k \times N_{val} \times cost\ of\ forward\ pass$. Assuming that the cost of a backward pass is roughly equal to twice the cost of a forward pass, the cost of computing a gradient-based saliency metric also using N_{val} batches is $3 \times N_{val} \times cost\ of\ forward\ pass$. The cost of the myopic oracle is hence not prohibitive but needs to be factored when choosing k. A wider view may yield better results but at an increased computational cost.

The myopic oracle visits each of the constituent saliency metrics in a round-robin fashion, and selects the lowest ranked channel to add to the set of channels whose sensitivity should be measured. If the lowest ranked channel has already been selected by another constituent, the second lowest is used instead, and so on. This process continues until k unique channels have been selected. The sensitivity of each channel is then tested, yielding the true ranking of these k channels.

Note that the actual saliency values output by each saliency metric are never consumed by the oracle, only the implied ordering of the channels is used. In this way, the oracle is agnostic to the scales of the individual pruning metrics.

Figure 4-2a the illustrates selection of channels to be evaluated by the oracle in the case of a pruning algorithm with $k = 3$. The selected channels then have their sensitivities measured by the oracle according to Equation 4.1 and Figure 4-2b.

4.3.2 Constituent Saliency Metrics

Our composite approach can be used with any saliency metric which can be expressed as a function of weights and feature maps (including all gradients, which are derivatives of one with respect to the other). However, composing all published saliency metrics following this schema would be unrealistic. Instead we choose a sample of prominent saliency metrics from the literature that perform well in practice. These metrics rely on different kinds of information. We consider weight-based, feature map-based, and gradient-based saliency metrics.

We selected the constituent saliency metrics shown in Table 4.1 to be combined using the myopic oracle. Prior work has shown each of these saliency metrics are very effective.

Even though they are known to perform well, the chosen saliency metrics are constructed under different assumptions and use a diverse selection of parameters from the network.

Saliency Metric	Equation
Mean squares of weights (Molchanov et al. 2017)	$${}^lS_i = \frac{1}{\|{}^lW_i\|_0} \sum_{w \in W_c} w^2 \tag{4.2}$$
Mean of activations (Anwar et al. 2017)	$${}^lS_i = \frac{1}{\|{}^lA_i\|_0} \sum_{a \in {}^lA_i} a \tag{4.3}$$
Average of gradients (Liu and Wu 2019)	$${}^lS_i = \frac{1}{\|{}^lA_i\|_0} \left| \sum_{a \in {}^lA_i} \frac{d\mathcal{L}}{da} \right| \tag{4.4}$$
1st order Taylor expansion (Molchanov et al. 2017)	$${}^lS_i = \frac{1}{\|{}^lA_i\|_0} \left| \sum_{a \in {}^lA_i} a \frac{d\mathcal{L}}{da} \right| \tag{4.5}$$
2nd order Taylor expansion using Fisher information (Theis et al. 2018)	$${}^lS_i = \frac{1}{2} \left(\sum_{a \in {}^lA_i} a \frac{d\mathcal{L}}{da} \right)^2 \tag{4.6}$$

Table 4.1: Notable saliency metrics used for channel pruning.

4.4 Experimental Setup

4.4.1 Saliency Metrics

For our experimental setup, we choose a set of constituent saliency metrics in Table 4.1 to compose via the myopic oracle, and also a general pruning algorithm to follow.

4.4.2 Hyperparameters

The myopic oracle is evaluated with $k = 5, 8, 12, 16$.

N_{val} is set to 2 for networks using the CIFAR-10 dataset and to 4 for networks using the CIFAR-100 dataset. The batch size used is 128. Hence, the total number of images, used for measuring the constituent saliency metrics and the oracle, is 256 for CIFAR-10 and 512 for CIFAR-100. It should be noted that even though the images for evaluating the saliency metrics and the oracle are randomly sampled at each iteration, within the same iteration the same images are used for computing the constituent saliency metrics and myopic oracle. This is done to avoid introducing more information available to the constituent metrics and oracle if more constituent metrics are used or for higher values of k.

The experiments are run 8 times and the average pruning rates over these 8 runs are presented in the results section along with their associated error bars.

4.4.3 Datasets and CNN Models

We run the experiments using the CIFAR-10 and CIFAR-100 datasets. The datasets are used as described in Section 3.8.3.

For the CIFAR-10 dataset, we use the following architectures: LeNet, CIFAR10, ResNet-20, NIN, and ResNet. For the CIFAR-100 dataset, we use the following architectures: ResNet-20, NIN, and AlexNet. The CNN models used are described in Section 3.8.4.

4.4.4 Pruning Algorithm

Since our objective is specifically to study the differences in pruning metrics, we choose to eliminate confounding factors by using a simple, iterative pruning algorithm without fine-tuning or retraining. The only change we make to the network weights is to set pruned weights to zero. Using a pruning retraining with retraining is needed to find the absolute best network, however introducing retraining introduces more stochasticity in the results and can compensate the poor choices of the saliency metric. Since our aim is not to find the best network but the best saliency metric, retraining can obfuscate the results. As seen in Section 3.9.5, even when retraining is in use, saliency metrics which cause less deviation from the initial test accuracy can lead to less time being spent on retraining, and also to large groups of channels being simultaneously removed, in the case of pruning algorithms that allow for simultaneous pruning of multiple channels. Hence, a better saliency metric will reduce the total amount of effort used to produce pruned networks. Algorithm 2 outlines the simple pruning algorithm used to evaluate the myopic oracle.

We use Algorithm 2 with $testAccDrop = 5\%$.

4.5 Results

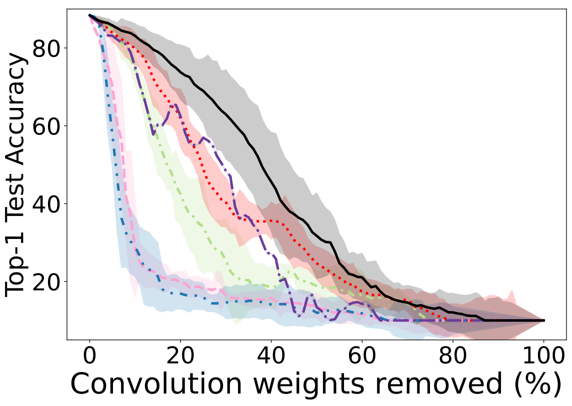
Saliency Metric		**LeNet-5**	**CIFAR10**	**ResNet-20**	**NIN**	**AlexNet**
Mean of activations		24±0.1	29±2	6±1	14±1	37±6
		-	-	2.6±0.5	38.1±0.2	22 ± 0.2
1st order Taylor expansion		33±6	16±0.1	2±0.1	14±1	49±6
		-	-	2.3±0.3	38.2±0.1	47 ± 0.4
2nd order Taylor expansion		22±5	39±2	6±3	21±2	43±6
		-	-	1.1±0.1	38.5±0.4	52 ± 6
Average of gradients		25±6	24±4	3±0.3	24±1	46±9
		-	-	2.1±0.3	36.7±0.3	50 ± 7
Mean squares of weights		17	23	5	28	49
		-	-	1.7	40.7	45
Myopic Oracle	$k = 5$	29±5	37±4	9±4	31±2	52±0.3
		-	-	3.2±0.3	39.5±0.4	55 ± 7
	$k = 8$	26±3	40±3	10±4	31±2	53±5
		-	-	3.4±0.3	39.8±0.4	57 ± 7
	$k = 12$	27±4	41±2	11±4	32±2	58±3
		-	-	3.7±0.4	39.9±0.4	60 ± 0.8
	$k = 16$	26±7	43±3	11±3	33±2	61±4
		-	-	3.7±0.3	40.0±0.6	60 ± 1

Table 4.2: Convolution weights (%) removed for a 5% accuracy drop on CIFAR-10 (grey) and CIFAR-100.

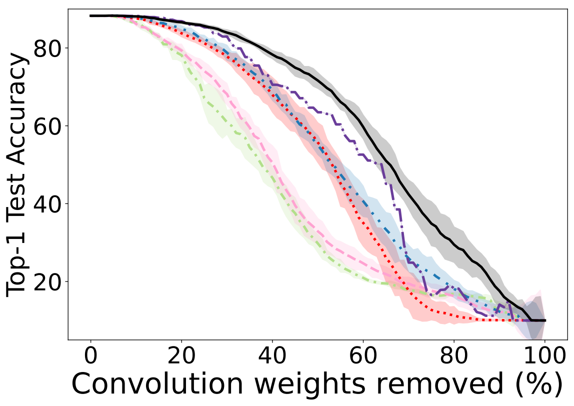
Figure 4-4 presents the result of our experimental evaluation on the five chosen convolutional neural networks. For all five networks, we see that the composite saliency metric matches or exceeds the predictive quality of any of the individual constituent metrics until the test accuracy of the network drops far below useful levels.

4.5.1 Behaviour of Composite Metrics

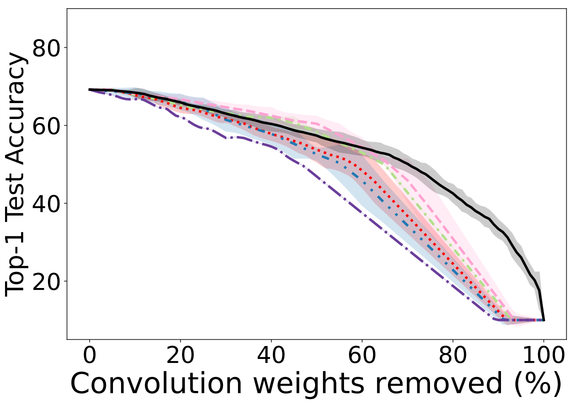
We would like to draw attention to the ResNet-20 (Figure 4-4a) and NIN (Figure 4-4b) networks in particular. For ResNet-20 (Figure 4-4a), our experiment shows clearly that some saliency metrics are very badly suited for guiding pruning on this



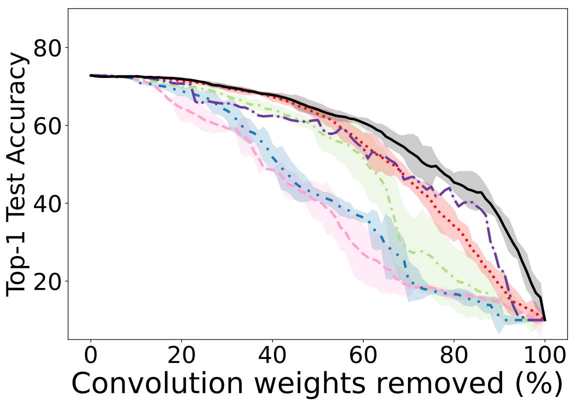
(a) ResNet-20



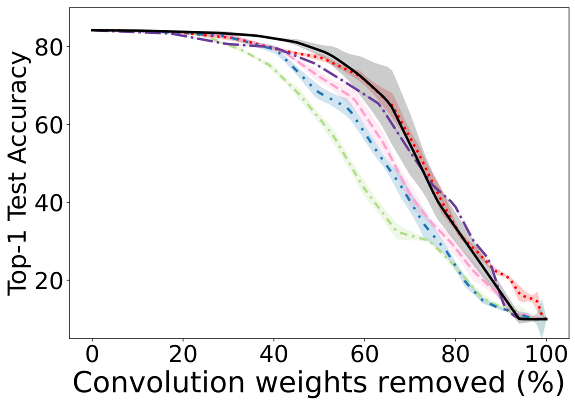
(b) NIN



(c) LeNet-5



(d) CIFAR10



(e) AlexNet

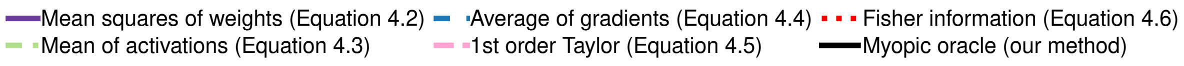


Figure 4-4: Graphs show top-1 test accuracy versus number of convolution weights (%) removed by pruning using the CIFAR-10 dataset. Individual saliency metrics are indicated with dashed lines, and the myopic oracle (with $k = 8$) is indicated with a solid line. Error bands for the myopic oracle are shown based on a 95% confidence interval for 8 runs of the experiment.

network. It is not that these are bad metrics; on the contrary, they perform well on other networks.

However, the assumptions baked into these metrics are at odds with the reality of the relationships of the weights, feature maps, and gradients in ResNet-20, causing them to severely mispredict the effect on the loss function of pruning any individual channel. Using the myopic oracle allows these metrics to be excluded until their assumptions become more in line with the reality of the network structure, instead of causing pathological behaviour if used indiscriminately.

For NIN (Figure 4-4b), our experiment shows that the composition of metrics via the oracle exhibits smooth, predictable behaviour, where the individual metrics differ dramatically.

Even though the individual metrics have such large differences, the composition of the metrics with the oracle is well-behaved, leading to a much less damaging pruning that with any of the metrics individually.

The remainder of the networks exhibit similar behaviour. For AlexNet (Figure 4-4e), we see again that the composition of the saliency metrics via the oracle yields a smooth, well-behaved metric, even though the constituent metrics have large differences.

4.5.2 Impact of k

From Table 4.2, we can see that using a myopic oracle can lead to a significant increase in the maximum number of weights but only to a marginal increase when increasing k. This trend would suggest that the channel rankings given by the individual saliency metrics are often accurate. Considering more channels only offers a marginal improvement as the least salient channels are often also ranked lowly by at least one of the constituent metrics, we only need to determine which saliency metric is accurate for that pruning iteration. Hence, choosing k to be equal to the number of constituent metrics allows us to choose between the saliency metrics without inhibitively increasing cost of computation.

4.5.3 Quality of Pruned Networks

Table 4.2 summarises the level of pruning achieved in our experiments for a maximum reduction of 5% points in top-1 test accuracy.

Using the myopic oracle to compose existing saliency metrics yields a composite metric which makes better pruning decisions than any of the individual metrics which were composed. The myopic oracle consistently selects channels to prune that result in a smaller loss in test accuracy. We also present the proportion of weights removed for the constituent saliency metrics, if used exclusively, as in prior work.

On every network, our approach meets or exceeds the performance of all the state-of-the-art saliency metrics used individually. The best results are seen on ResNet-20, where almost twice as many weights can be removed using our approach versus the next-best individual saliency metric.

4.6 Discussion

4.6.1 Pruning Algorithm

Combining saliency metrics with a myopic oracle within a simple pruning algorithm yields promising results. However, more sophisticated saliency-based pruning algorithms can also take advantage of using multiple saliency metrics to remove the maximum number of weights. Our future work will cover testing the use of multiple saliency metrics using a myopic oracle in other pruning algorithms.

4.6.2 Other Granularities

The use of multiple saliency metrics is not limited to channel pruning, it can apply to other granularities of pruning. For example, with unstructured pruning (fine-grain pruning) each saliency metric proposes a different set of weights to be removed. The myopic oracle can then choose which proposed set is most accurate.

4.7 Concurrent Related Work

The use of multiple saliency metrics by a single pruning algorithm is a new technique.

Concurrently to our work on composition of saliency metrics, He et al. (2020) also proposed a method using multiple saliency metrics within a single pruning algorithm. He et al. (2020) propose a one-shot pruning algorithm that automatically selects a different saliency metric (referred to as pruning criteria) for each layer of the network. Prior to the pruning process, they learn the probabilities of pruning a given layer by a given saliency metric. The total number of probabilities that they learn is given by the number of layers times the number of saliency metrics in their search space. They learn these probabilities by minimising the loss of the network using the sum of the pruned feature maps (i.e. feature maps pruned according to each metric) instead of the unpruned feature maps. Finally, they select the criteria with the highest probability per layer and proceed with a standard pruning-retraining phase.

In contrast, our method uses multiple saliency metrics (chosen by us) to create a smaller search space for the myopic oracle. The myopic oracle then evaluates the sensitivity of the chosen candidates to provide a composite metric. The composite metric can be used to substitute any saliency metric in a saliency-based pruning algorithm.

Another method using saliency metrics to create a smaller search space, is the recently published work by Zhang et al. (2021). Instead of using a genetic algorithm on the entire channel pruning search space, they restrict the search space using a chosen number of saliency metrics. They then use a genetic algorithm to explore the possible pruning configurations.

4.8 Conclusion

Our method of composing multiple saliency metrics yields a composite metric that significantly outperforms the individual constituent metrics. By dynamically switching between different metrics based on the actual measured sensitivity of the network, we avoid the occasional poor pruning decision made by even the most advanced saliency metrics. Using our approach, data scientists are freed from having to choose from a dizzying array of potential saliency metrics to guide the pruning process.

By developing a method to dynamically switch between an arbitrary collection of state-of-the-art saliency metrics based on their actual measured performance, we can derive a composite metric with significantly improved performance, pruning up to twice as many weights for the same drop in accuracy in our experiments. Our approach advances the state-of-the-art in identifying unnecessary or redundant sets of neural network parameters.

Chapter 5

Domino Saliency Metrics: Improving Existing Channel Saliency Metrics with Structural Information

Channel pruning removes slices of the weight tensor so that the convolution layer remains dense. The removal of these weight slices from a single layer causes mismatching number of feature maps between layers of the network. A simple solution is to force the number of feature map between layers to match through the removal of weight slices from subsequent layers. This additional constraint becomes more apparent in CNNs with branches where multiple channels need to be pruned together to keep the network dense. Popular pruning saliency metrics seen in Chapter 3 do not factor in the structural dependencies that arise in CNNs with branches.

In this chapter, we propose *domino* metrics (built on existing channel saliency metrics) to reflect these structural constraints. We test domino saliency metrics against the baseline channel saliency metrics on multiple networks with branches. Domino saliency metrics improved pruning rates in most tested networks and up to 25% in AlexNet on CIFAR-10.

5.1 Introduction

Channel pruning removes weights corresponding to an entire channel in the output of a layer. When the weights of the entire channel are set to zero, the corresponding

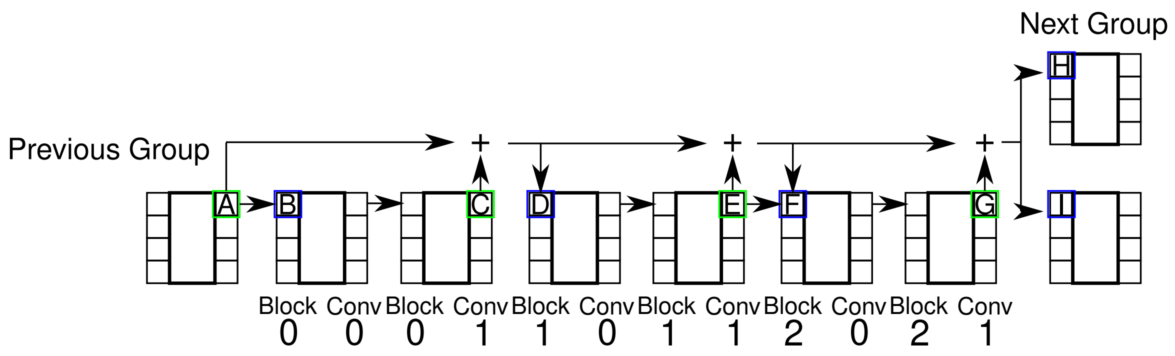


Figure 5-1: Structure of a block in ResNet-20. The arrows are in the direction of data flow.

output feature map of the channel becomes zero. This zero output feature map feeds into subsequent layers which may allow the corresponding channel to be removed from these subsequent layers in a *domino effect*.

In a network architecture where each layer has exactly one output and one input layer, the removal of an output feature map leads to the following layer's input feature map no longer contributing to the network. This scenario is illustrated in Figure 5-5 where the consumer layer is a either convolution or fully-connected layer, the resulting zero input feature map allows the corresponding weights from the consumer layer to be pruned.

In networks with splits (commonly called branches in the context of CNNs), the output feature map from one layer may feed into multiple layers. With joins, feature maps from different layers feed into a single layer. A common occurrence of split and join connections in CNNs is due to skip connections, which were pioneered in ResNet architectures (He et al. 2016). The common structure of a ResNet block containing split and join connections is shown in Figure 5-1. The presence of these splits and joins allows multiple output feature maps to be removed together when one feature map is considered for removal. Networks that offer state-of-the-art accuracy for image classification often contain skip connections (Brock et al. 2021; Zhang et al. 2020; Xie et al. 2016; Mahajan et al. 2018; Touvron et al. 2019). ResNet architectures are also more difficult to prune for their lower redundancy (Luo et al. 2017; Dong et al. 2017b; He et al. 2017). Hence, improving pruning rates for networks containing skip connections can be very advantageous.

Another common occurrence of joins is group convolution. Group convolution was originally used in the AlexNet architecture (Krizhevsky et al. 2017) to parallelise convolution on multiple GPUs. Since, it has also been used in state-of-the-art architectures (Xie et al. 2016; Brock et al. 2021).

5.2 Contribution

Most approaches do not factor in the removal of different feature maps when computing the saliency metric used for pruning. In this chapter, we propose using *domino saliency metrics* to factor in the saliency of feature maps and weights that need to removed together. We make the following contributions:

- We propose domino saliency metrics, where existing saliency metrics are combined together depending on the set of channels that need to be removed together.
- Combining channel saliency metrics to obtain domino saliency metrics, has a negligible computational cost to computing channel saliency metrics and requires very little modification to existing pruning strategies.
- We experimentally evaluate two variants of domino saliency metrics: *Domino-o* and *Domino-io*, and find that they significantly improve pruning.

5.3 Data Flow Graph for Pruning

5.3.1 Channel Pruning Networks with Splits and Joins

Channel pruning removes entire channels from convolution layers of the network. By removing entire channels, the weight tensor remains dense, so existing DNN dense libraries can be used. The removal of an entire channel of convolution weights leads to the removal of its subsequent feature map. Removing a feature map may allow corresponding weights from subsequent layers to be removed. In a network where each layer has at most one input layer and one output layer, this relationship is obvious.

While simpler neural networks are often linear and acyclic (Lecun et al. 1998; Krizhevsky 2009), modern networks often contain join nodes and split nodes (Krizhevsky et al. 2017; He et al. 2016).

It is obvious which weights need to be removed when applying channel pruning to a network where each layer is fed to only one successor. However in networks with branches, feature maps are used by more than one layer. A layer can have multiple successors due to skip connections. Skip connections are element-wise additions between output feature maps of different convolutions to produce the input feature map of the following layer.

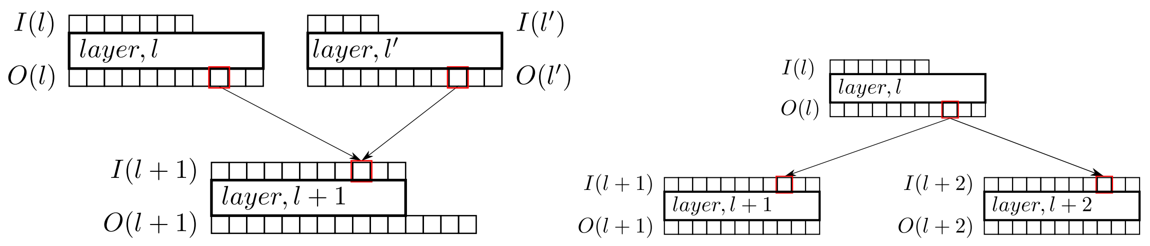
In networks with branches, we can perform a reachability analysis, such as is performed by a compiler on a more traditional computational structure, the control-flow graph, to uncover the weights that need to be pruned together to keep the network dense. The basic intuition here is that if the definition of some feature map reaches a layer, the pruning of that feature map may also imply the removal of more feature maps which are computed from it. While this seems trivial for linear networks, the introduction of splits and joins in the graph mean that extra care must be taken in order to exploit the dependence relationship to achieve better pruning results.

5.3.2 Data Flow Graph

Neural networks form a directed graph structure where simple input-output dependence exists between producer and consumer layers in the network (i.e. the network forms a *data flow graph*). When representing the data flow graph, we can make abstraction of activation layers. This abstraction is discussed in details in Section 5.3.6. Hence, we only need to represent the data flow between a convolution or fully-connected layer to other convolution or fully-connected layers.

We introduce additional notation to facilitate the description of the data flow graph.

A layer l has a set of 3-dimensional input feature maps $I(l)$, a set of 3-dimensional output feature maps $O(l)$ and weights ${}^{l}W$. $O_i(l)$ and $I_i(l)$ refer to the i^{th} 2-dimensional feature map of $O(l)$ and $I(l)$ respectively. In our previous notation we had $O(l) = {}^{l}A$. For simple forward feed networks with no branches, we had $I(l) = {}^{l-1}A$.



(a) Join connection in CNN. (b) Split connection in CNN.

Figure 5-2: Data dependencies with join and split connections in CNNs. The arrows are in the direction of data flow.

We can describe the flow of information between layers with a successor relation $O(l) = I(succ(l))$. Intuitively, the data flow successors, $succ(l)$ of a layer l are those layers whose input is the output of layer l.

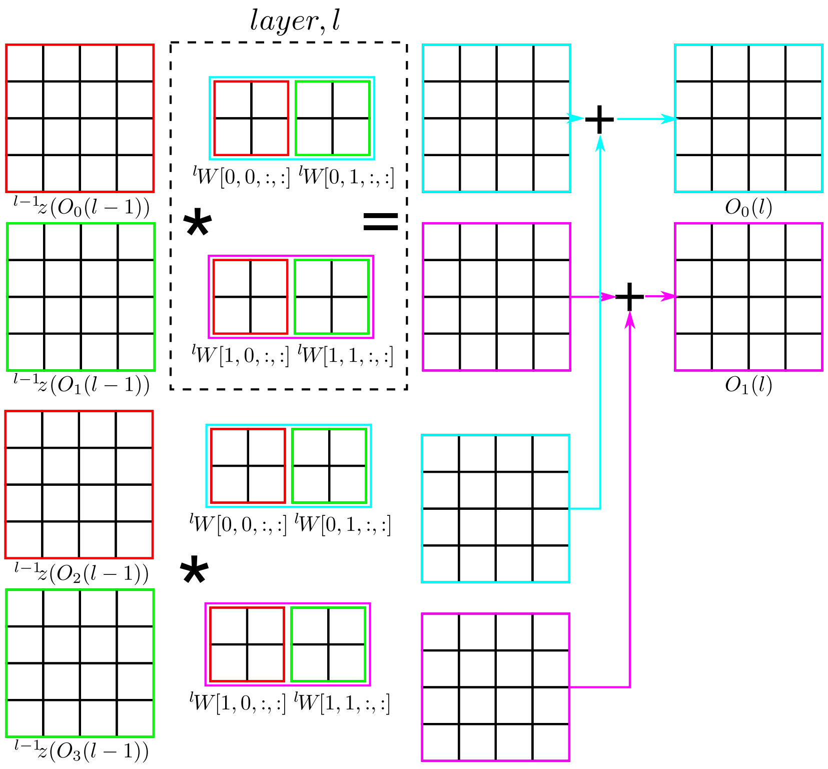
When a layer in a neural network joins the output of multiple producer channels, we write $O(l) \subset I(succ(l))$ i.e. the successor relation extends to any channel which consumes the output of l. If $I(l + 1)$ is the input feature map of the following, then it is the direct successor of $O(l)$.

5.3.3 Join and Split Nodes

$I(l + 1)$ is not necessarily the only successor of $O(l)$. For example, in Figure 5-2a the element-wise sum of two layers l and l' is fed to layer $l + 1$. The layer $l + 1$ satisfies $O(l) \subset I(l+1) \wedge O(l') \subset I(l+1)$. We can also consider data dependencies for a single channel, ${}^{l}C_i$, instead of an entire layer, l, with $O({}^{l}C_i) = O_i(l)$. So we would say that both $succ({}^{l}C_i) = \{{}^{l+1}C_i\}$ and $succ({}^{l'}C_i) = \{{}^{l+1}C_i\}$. Split, or broadcast relationships also exist, where the output of a layer is used by multiple consumers. An example of split in the data flow graph is seen in Figure 5-2b where $succ({}^{l}C_i) = \{{}^{l+1}C_i, {}^{l+2}C_i\}$.

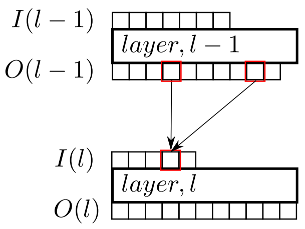
5.3.4 Group Convolution

Data dependencies at the entrance of group convolutions can be modelled using a simple convolution layer that has g sets of input feature maps that result in g sets of partial output feature maps. These partial output feature maps are then added together to create the final output feature map of the layer. We have ${}^{l+1}m_{in} = \frac{{}^{l}m_{out}}{{}^{l}g}$ with the group size, ${}^{l}g > 1$. Figure 5-3 illustrates a group convolution with a group size of 2.



$layer, l$
$^{l-1}z(O_0(l-1))$
$^{l-1}z(O_1(l-1))$
$^{l-1}z(O_2(l-1))$
$^{l-1}z(O_3(l-1))$
$^lW[0,0,:,:]$ $^lW[0,1,:,:]$
$^lW[1,0,:,:]$ $^lW[1,1,:,:]$
$^lW[0,0,:,:]$ $^lW[0,1,:,:]$
$^lW[1,0,:,:]$ $^lW[1,1,:,:]$
$O_0(l)$
$O_1(l)$

(a) Group convolution with $^lg = 2, {}^lm_{in} = 2, {}^{l-1}m_{out} = 4$.



(b) Group convolution data dependency.

Figure 5-3: Data flow with group convolutions in CNNs. The arrows are in the direction of data flow.

For group convolutions, $O(^{l-1}C_{i+j\times^l m_{in}}) \subset I(^lC_i)$, for $j \in 0...^lg-1$. In Figure 5-3b, $succ(^{l-1}C_i) = \{^lC_i, {}^lC_{i+2}\}$.

5.3.5 Channel Pruning

Considering a neural network graph with G channels, we can define the successor relation as:

$$\forall c \in G, \exists x \in G : O(c) \subset I(x) \Rightarrow x \in succ(c) \qquad \text{(SUCCESSOR)}$$

Let the predicate $P(c)$ be true where a feature map c is being pruned, and false otherwise. The truth of $P(c)$ is determined locally for the input or output feature maps of a specific layer by the pruning process.

We are interested in how the truth of $P(c)$ locally in any single layer may influence the truth of $P(c)$ for connected layers in the network, or more informally, how removing some feature maps may imply the removal of other feature maps.

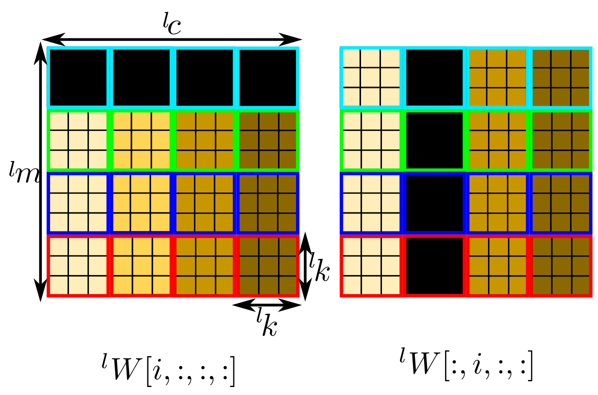


Figure 5-4: Pattern of weights removed when removing output (left) and input (right) channels. The dark squares represent pruned parameters.

$$\forall c \in G, \forall x \in succ(c) : P(O(c)) \Leftrightarrow P(I(x)) \qquad \text{(CHANNEL PRUNING)}$$

Equation CHANNEL PRUNING states that the pruning an output feature map $O(c)$ is materially equivalent to pruning the input feature map $I(x)$, with x a successor of c. This predicate is valid for all the channels in the network. Hence, in a typical ResNet style block this leads to the simultaneous pruning of multiple channels. In Figure 5-1, feature maps A to I need to be removed simultaneously.

Equation CHANNEL PRUNING is used to propagate pruning of feature maps. However, for channel pruning, we need to remove weights from the network. To prune an output channel, $c = {}^lC_i$, from the network, weights ${}^lW[i,:,:,:]$ are removed from network with ":" representing all valid indices. Hence, when an output feature map $O({}^lC_i)$ is pruned, an output channel ${}^lW[i,:,:,:]$ is pruned and when an input feature map $I({}^lC_i)$ is pruned, an input channel ${}^lW[:,i,:,:]$ is pruned. The structures of the weights removed are illustrated in Figure 5-4.

The set of weights from the network removed when a channel ${}^lC_i$ is called $\mathcal{TC}({}^lW_i)$.

5.3.6 How to Prune Biases and Activation Layers

The output of a convolution is not often fed directly to the next convolution or fully-connected layer. Instead, at least one activation function is applied to the feature map before being fed to the next layer. Hence, the output feature map produced by a layer is not directly equivalent to the input feature map of the following layer.

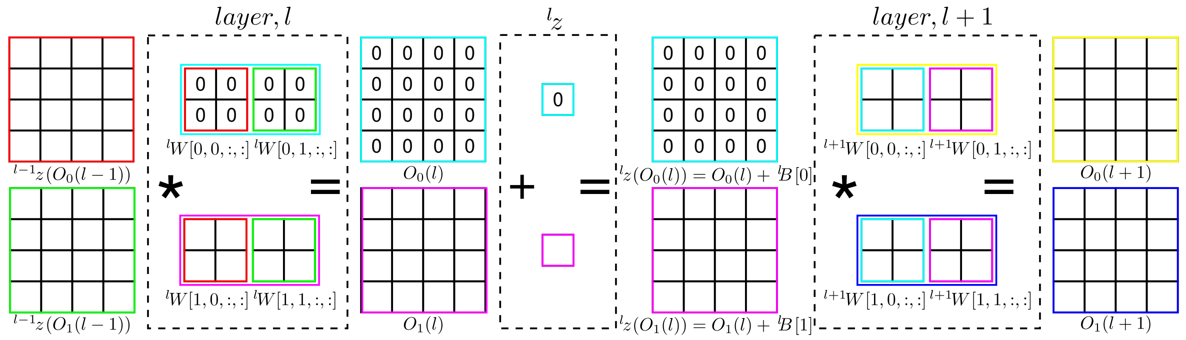

$layer, l$
^{l}z
$layer, l+1$
$^{l-1}z(O_0(l-1))$
$^{l-1}z(O_1(l-1))$
$^{l}W[0,0,:,:]$ $^{l}W[0,1,:,:]$
$^{l}W[1,0,:,:]$ $^{l}W[1,1,:,:]$
$O_0(l)$
$O_1(l)$
0
$^{l}z(O_0(l)) = O_0(l) + {}^{l}B[0]$
$^{l}z(O_1(l)) = O_1(l) + {}^{l}B[1]$
$^{l+1}W[0,0,:,:]$ $^{l+1}W[0,1,:,:]$
$^{l+1}W[1,0,:,:]$ $^{l+1}W[1,1,:,:]$
$O_0(l+1)$
$O_1(l+1)$

Figure 5-5: Channel pruning where the outputs of convolution l are fed into an activation layer (with bias), ^{l}z before being fed to the successor $l+1$.

For most activation functions, applying them to a zero feature map results in a zero feature map. ReLU, GELU, and max/average pooling layers, are examples of activation functions that output zero for a zero input. When these activation functions are used, a pruned output feature map results in pruned input feature maps of the subsequent layers. However, in the case of activation functions with biases such as Batch Norm or bias layers, a zero input does not always result into a zero output unless the biases are also set to zero. Hence, the pruned feature map and the pruned weights cannot be removed from the network. However, if the corresponding biases are set to zero, the weights can then be removed from the network.

A simple channel pruning case is shown in Figure 5-5. For the convolution layer l, to produce a single $O_i(l)$ each j^{th} input feature map is convolved with its corresponding 2D filter $^{l}W[i,j,:,:]$ and summed together. In Figure 5-5, $O_0(l) = {}^{l-1}z(O_0(l-1)) * {}^{l}W[0,0,:,:] + {}^{l-1}z(O_1(l-1)) * {}^{l}W[0,1,:,:]$ and $O_1(l) = {}^{l-1}z(O_0(l-1)) * {}^{l}W[1,0,:,:] + {}^{l-1}z(O_1(l-1)) * {}^{l}W[1,1,:,:]$. To prune the output channel $^{l}C_0$, $^{l}W[0,0,:,:]$ and $^{l}W[0,1,:,:]$ are set to zero. If the corresponding bias in the activation layer ^{l}z ($^{l}B_0$), is set to zero, then the input feature map $^{l}z(O_0(l))$ is also zero. Since convolution with a zero feature map results into a zero feature map, the values of $^{l+1}W[0,0,:,:]$ and $^{l+1}W[1,0,:,:]$ no longer influence feature maps $O_0(l+1)$ and $O_1(l+1)$. Hence, $^{l}W[0,0,:,:]$, $^{l}W[0,1,:,:]$, $^{l}B_0$, $^{l+1}W[0,0,:,:]$, and $^{l+1}W[1,0,:,:]$, can be set to zero and removed.

If the biases are not set to zero, then feature maps filled with zeros still need to be stored to keep the network dense. In practice, output feature maps filled with zeros are removed from the network to increase memory savings. Hence, biases of activation functions are also set to zero and the obsolete parameters are removed to keep the network dense. To simplify the data flow graph, we can make abstraction of activation functions even if they contain biases. We can also assume that when a channel is pruned, weights, feature maps, and activation layer parameters of that channel are set to zero.

5.4 Domino Pruning

When pruning neural networks with skip or group connections, most approaches use the same saliency metrics as for simple forward feed neural networks without any modification. The construction of these saliency metrics do not take into account the structural dependencies that may to be satisfied between layers to keep the network dense.

We argue that a more straightforward inspection of the data dependence structure may be more prudent. Using a reachability analysis such as may be performed by a compiler, we show how the removal of one set of parameter may be used to heuristically perform a cascade of removals of other reachable parameters. We refer to this technique as "domino" saliency metrics. Any saliency metric that has been formulated to give each channel a saliency measure can be used with domino saliency metrics.

$$\forall x \in succ(c), \exists y : x \in succ(y) \Rightarrow y \in coparent(c) \tag{COPARENT}$$

$$\forall x \in succ(c), \exists y : y \in succ(c) \Rightarrow y \in sibling(x) \tag{SIBLING}$$

With domino pruning, the output channels that are considered coparents are considered as a single pruning choice. Hence, they cannot be removed without the other coparent output channels. Output channels are considered coparents if they share a common successor as shown in Equation COPARENT. Similarly Equation SIBLING is used to find siblings of direct successors. We consider the *coparent* and *sibling* relationships to be birectional and transitive, i.e., if $A \in coparent(B)$ then $B \in coparent(A)$, and if $C \in coparent(B)$ then $C \in coparent(A)$. The transitive closure of $coparent$ and $sibling$ are respectively called $coparents^{+}$ and $siblings^{+}$. The feature maps and weights removed when one output channel is removed is given by Equation CHANNEL PRUNING. When a channel c is pruned, output feature maps from the set $coparents^{+}(c)$ and their associated output channel weights are pruned. Input feature maps from the set $siblings^{+}(c)$ and their associated input channel weights are also pruned when channel c is removed. When c is an output channel, then $siblings^{+}(c)$ represents the transitive closure of any of c's successors and when c is an input channel, $coparents^{+}(c)$ represents the transitive closure of any output channel that is used to produce c.

The set of feature maps pruned is $coparents^{+}(c) \cup siblings^{+}(c)$. In the case of a network with no joins and no splits, $coparents^{+}(c) = \{c\}$ and $siblings^{+}(c) = succ(c)$. In the ResNet block shown in Figure 5-1, $coparents^{+}(A) = \{A, C, E, G\}$ and $siblings^{+}(A) = \{B, D, F, H, I\}$.

Saliency metrics are traditionally computed for a single output channel. This is also true for networks with skip connections (ResNets). With domino metrics, we combine the channel saliency of channels that are removed together.

We propose two variants of domino saliency: *Domino-o* and *Domino-io*.

Domino-o adds the saliency of all output channels that are removed together. Since most channel pruning algorithms use the channel saliency of output channels, *Domino-o* has negligible cost and require minimal change to the pruning frameworks. Equation 5.1 decribes how to compute *Domino-o* for a channel using $coparents^{+}$ and their channel saliency, S. As seen in Chapter 3, the channel saliency can be obtained using the feature or its channel weights.

$$Domino\text{-}o(c) = \sum_{x \in coparents^{+}(c)} S(x) \tag{5.1}$$

As illustrated in Figure 5-5, the output feature produced by a convolution channel ultimately becomes the input feature map for the following layer. Few approaches, apart from feature reconstruction-based metrics, exploit this relationship for saliency computation. With *Domino-io*, we add the saliency of all the weights or feature maps that are removed when a channel is removed to get the saliency of the channel to be pruned. Equation 5.2 shows how to compute *Domino-io* using the channel saliency of $coparents^+$ (output feature maps or weights) and $siblings^+$ (input feature maps or weights) using a channel saliency S.

$$Domino\text{-}io(c) = \sum_{x \in coparents^+(c)} S(x) + \sum_{x \in siblings^+(c)} S(x) \tag{5.2}$$

5.5 Experimental Evaluation

Channel saliency is the use of the saliency metric of a single output channel to prune all the dependent weights and feature maps. This is the common strategy when pruning networks. In practice, this leads to the lowest saliency of the output channels that are pruned together to be used as saliency metric of the set of weights or feature maps to be pruned. This baseline is denoted S.

We compare domino saliency metrics constructed using a channel saliency against the use of the base channel saliency.

A domino saliency metric is built using a baseline channel saliency metric. If the saliency of all channels are roughly of the same order and positive, their addition is of greater order than channels that are not part of split or join nodes. We use an average of the number of weights or output points to avoid favouring isolated channels for pruning. Saliency metrics that use feature maps are scaled using the number of pixels in the feature maps. For example, if two saliency values, $S(c)$ and $S(z)$, are computed using N_c and N_z weights then, the scaled domino metric is $\frac{S(c)+S(z)}{N_c+N_z}$ instead of $S(c) + S(z)$. The scaled channel metrics are $\frac{S(c)}{N_c}$ and $\frac{S(z)}{N_z}$. Metric with this average is suffixed with -avg.

5.5.1 Pruning Algorithm

The saliency metric is one component of the pruning algorithm. We choose a pruning algorithm that heavily relies on the choices of the saliency metric to determine the improvement brought by changing the saliency metric. Since our aim is to find better saliency metrics, we avoid obfuscating the contribution of the saliency metric by not retraining after each pruning step. As seen in Chapter 3, if a saliency metric is able to achieve higher pruning rates without retraining, then it can also be used to reduce the cost of retraining.

For each pruning iteration, we compute the saliency (either the channel/baseline saliency or the domino saliency metric) of every output channel. The lowest saliency channel is then pruned according to Equation CHANNEL PRUNING. This process is repeated until the test accuracy falls under 5% of its initial value. We use Algorithm 2 with $testAccDrop = 5\%$

5.5.2 Networks

We evaluate domino saliency metrics on popular architectures with split and join nodes. We evaluate the original ResNet architecture (He et al. 2016) and a state-of-the-art ResNet-inspired architecture, NFNET-F0 (Brock et al. 2021). We also evaluate domino saliency metrics on AlexNet (Krizhevsky et al. 2017). The join and split connections in ResNet arise due to skip connections. The join connections in AlexNet arise due to group convolutions. NFNET-F0 contains both skip connections and group convolutions.

We use the CIFAR-10 (Krizhevsky 2009), CIFAR-100 (Krizhevsky 2009), ImageNet-32 (Chrabaszcz et al. 2017) (a downsized ImageNet variant), and ImageNet (Deng et al. 2009) datasets. ResNet-20 [1] on ImageNet, ResNet-50 (Simon et al. 2016) on ImageNet and NFNET-F0 (Brock et al. 2021) on ImageNet are pretrained networks. The accuracy of the pretrained networks are given in Table 5.1.

	ResNet-20	**ResNet-50**	**NFNET-F0**
ImageNet	68.0%	75.0%	75.0%

Table 5.1: Summary of accuracy of pretrained networks on ImageNet.

The remaining networks are trained from scratch and presented in Section 3.8.4.

5.5.3 Hyperparameters

The number of images in a batch, B_{val}, and number of batches, N_{val}, used to evaluate the saliency metrics are given in Table 5.2.

[1]https://github.com/HolmesShuan/ResNet-18-Caffemodel-on-ImageNet

Network	B_{val}	N_{val}
CIFAR-10 dataset		
ResNet-20	128	2
AlexNet	128	2
CIFAR-100 dataset		
ResNet-20	128	2
AlexNet	128	2
ImageNet dataset		
ResNet-20	32	8
ResNet-50	32	8
NFNET-F0	32	8
ImageNet-32 dataset		
AlexNet	128	8

Table 5.2: Hyperparameters used to measure the base channel saliency metrics.

5.5.4 Saliency Metrics

The most popular pruning saliency metrics are either a derivative of the L1 or L2 norm of weights (Han et al. 2015; Han et al. 2017; He et al. 2018b; Frankle and Carbin 2019; Guo et al. 2016; Mao et al. 2017; Li et al. 2017a; Wang et al. 2018; Lebedev and Lempitsky 2016; He et al. 2018a) or Taylor expansions (LeCun et al. 1989; Hassibi and Stork 1992; Molchanov et al. 2017; Molchanov et al. 2019; Peng et al. 2019; Ding et al. 2019). With channel pruning, Taylor expansions can be applied to either weights or feature maps.

The channel saliency metrics that we evaluate are: Taylor expansion using weights, Taylor expansion using feature maps, and L1 norm of weights.

5.6 Results

We measure the improvement of domino saliency metrics by comparing the percentage of convolution weights removed for a drop of 5% in test accuracy. The results are an average from 4 runs.

Figures 5-6, 5-7, and 5-8 show the percentage of convolution weights removed. In most cases, we observe an improvement by using the domino saliency metrics. *Domino-io* significantly improves the pruning rates for AlexNet on CIFAR-10, AlexNet on ImageNet-32, and ResNet-20 on CIFAR-10. *Domino-io* includes the saliency of input feature maps or input channel weights in addition to *Domino-o*.

The larger improvement brought by *Domino-io* suggests that the saliency of input channels weights contain relevant information for pruning.

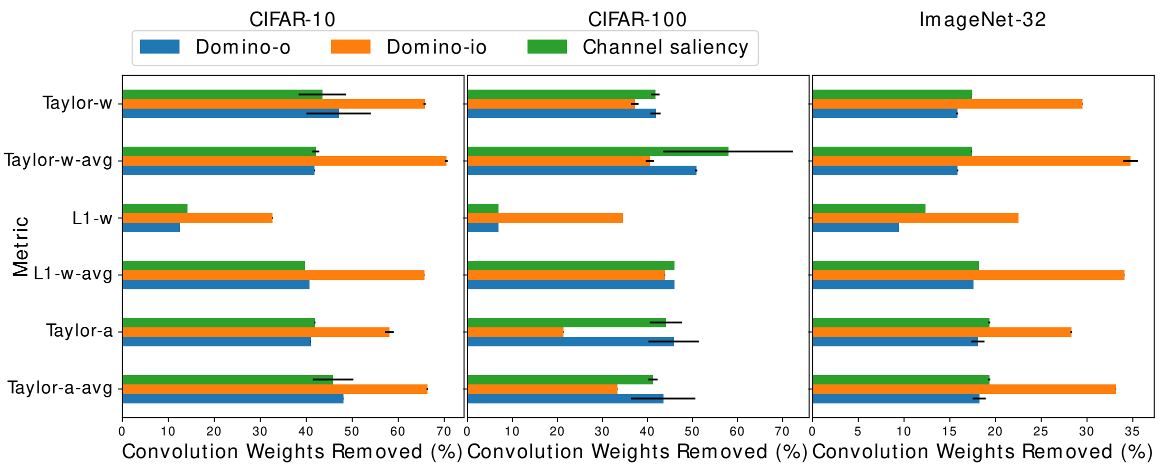


Figure 5-6: Domino saliency metrics results for AlexNet. Sparsity levels (percentage of convolution weights) are given on the x-axis. The results are grouped (by groups of three) according to the base metric given on the y-axis. The results of the base metric, *Domino-o*, and *Domino-io* are represented by the blue, orange and green bars respectively.

From Figure 5-6, we see that using *Domino-io* on AlexNet for CIFAR-10 and ImageNet greatly improves the base saliency metric. A notable result is the L1 norm of weights (with averaging) which can match the pruning rates of Taylor expansion-based methods with *Domino-io*. The L1 norm of weights is a very popular metric for pruning for its low computational cost and good pruning rates. *Domino-io* has a negligible cost overhead while greatly improving the L1 norm of weights (with averaging).

From Figure 5-7, we observe a similar trend where the L1 norm of weights (with averaging) can be improved to match and exceed the pruning rates of Taylor expansion-based method on ResNet-20 on ImageNet.

Figure 5-8 shows that the improvement of domino saliency metrics are marginal on NFNET-F0. However, since the pruning rates are also extremely low, the results are inconclusive for this network. On the other hand, ResNet-50 benefits from the additional structural information used by domino saliency metrics. The pruning rates on ResNet-50 can be improved by a few percentage points with either *Domino-o* or *Domino-io*.

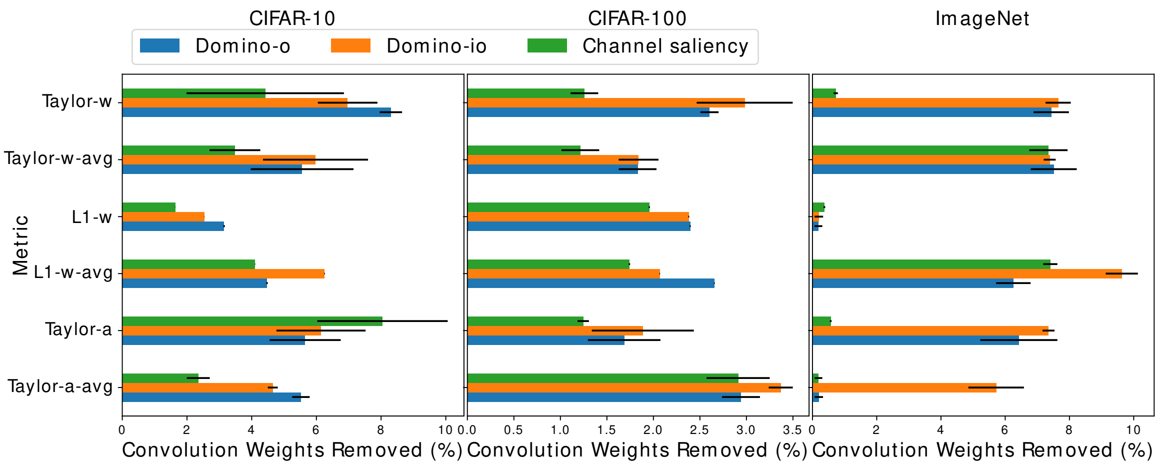


Figure 5-7: Domino saliency results for ResNet-20. Sparsity levels (percentage of convolution weights) are given on the x-axis. The results are grouped (by groups of three) according to the base metric given on the y-axis. The results of the base metric, *Domino-o*, and *Domino-io* are represented by the blue, orange and green bars respectively.

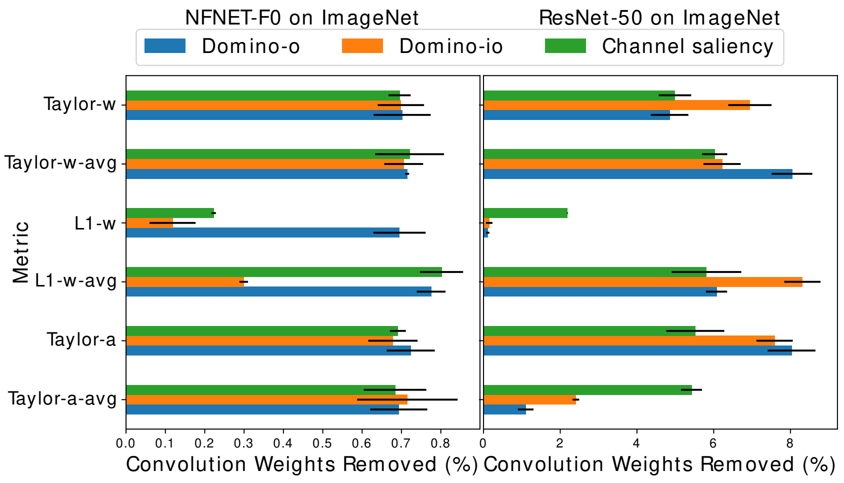
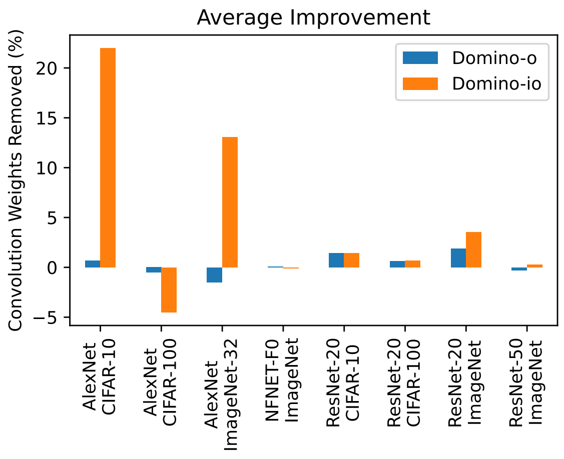
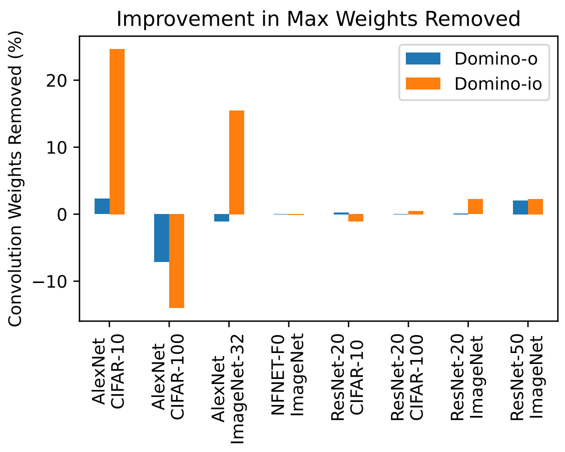


Figure 5-8: Domino saliency results for NFNET-F0 and ResNet-50. Sparsity levels (percentage of convolution weights) are given on the x-axis. The results are grouped (by groups of three) according to the base metric given on the y-axis. The results of the base metric, *Domino-o*, and *Domino-io* are represented by the blue, orange and green bars respectively.

The average improvement of using domino metrics over channel metrics for each network, is shown in Figure 5-9a. On all the networks, except AlexNet on CIFAR-100, *Domino-io* on average improves the baseline channel metric. *Domino-io* can be used to push the pruning rate of a network further. The improvement shown in Figure 5-9b corresponds to the difference between the maximum percentage of weights removed between the best domino metric and the best channel metric for a given network. With *Domino-io*, up to 25% and 15% more weights can be respectively removed from AlexNet on the CIFAR-10 dataset and the ImageNet-32 dataset.



(a) Average improvement.



(b) Improvement in maximum weights removed.

Figure 5-9: The improvement of domino saliency metrics over the base channel metric.

5.7 Related Work

When pruning networks with branch connections, multiple channels are removed to keep the network dense. However, most channel pruning approaches use the same saliency metric (Molchanov et al. 2019; Mao et al. 2017) as for simple forward feed networks. These saliency metrics do not factor in the other output or input channels that need to be removed when one channel is removed. Some pruning approaches that are based on feature-map reconstruction (He et al. 2017; Luo et al. 2019) explicitly describe how branches are taken in account. He et al. (2017) and Luo et al. (2019) remove output channels by considering their effect on the next layer's input feature map. For the ResNet architecture, He et al. (2017) consider the input feature map after the skip connection. Luo et al. (2019) avoid pruning layers that could result in mismatching number of feature maps in the network. Feature maps reconstruction-based approaches are computationally expensive. They are poor candidates for elaborate pruning algorithms (He et al. 2018b; He et al. 2018a; Wang et al. 2020b) that rely on computationally cost effective saliency metrics.

5.8 Conclusion

Most popular channel saliency metrics do not take into account structural constraints that can arise when pruning join and split connections. We propose two domino saliency metrics to add structural information to the saliency measure by combining channel saliency of multiple channels. *Domino-o* adds information about the other output channels that are removed together and *Domino-io* adds information about output and input channels that are removed together. We observe a small improvement when using *Domino-o* over the baseline channel saliency metric and a significant improvement when using *Domino-io*. *Domino-io* can be use to improve the pruning rates by 25% and 15% for AlexNet on CIFAR-10 and ImageNet-32. In conclusion, we find that the inclusion of the input channel saliency in *Domino-io* significantly improves pruning rates in networks with splits and joins.

Chapter 6

Future Work

6.1 Standard Benchmarking Setups for Pruning

Comparing pruning algorithms is difficult due to differing networks, points of convergence of these networks, datasets, goals, or even different ways of quantifying the pruned network (Blalock et al. 2020). Comparing saliency metrics is even more difficult as the saliency metric is one of the multiple aspects of pruning algorithms (Menghani 2021; Xu et al. 2020; Liu et al. 2020a; Blalock et al. 2020). In our experiments, we fix these aspects to isolate the effect of the saliency metric.

The lack of a standard benchmarking setup makes it impossible to directly compare reported pruning rates from different research papers. Even though pruning algorithms are often tailored to solve a specific problem, the existence of standard benchmarking setups would allow for easier comparison between pruning algorithms. Standard benchmarking setups would be a valuable contribution to the machine learning community.

6.2 Relevance of Pruning Trained Networks

In Chapters 3, 4, and 5, we evaluate saliency metrics using a classic pruning algorithm that starts with a trained network, iteratively prunes, and retrains the pruned network. Recent work has shown that pruning can be done before training (Frankle and Carbin 2019; Lee et al. 2019). Liu et al. (2019b) show that training the pruned structures from scratch can achieve similar accuracy levels to a pruned network obtained from the classic iterative pruning-retraining algorithm. With pruning not requiring trained weights, one may wonder if pruning of trained networks is still relevant.

In fact, pruning trained networks is still the most cost efficient solution when a large trained network is already available. Published CNNs that achieve 90% classification accuracy use billions of parameters (Zhai et al. 2021; Pham et al. 2021; Riquelme et al. 2021). Pruning trained networks is still a valid choice to find efficient high accuracy networks. Networks can be trained without a specific memory constraint and pruned to meet the memory constraints if they are to be deployed on devices with different memory resources. The cost of training is then a one-time expense with each pruned network requiring only the additional cost of weight removal and retraining. However, since channel pruning can be conducted at initialisation (Lee et al. 2019), a possible extension to our work would be investigating if our improvements also apply to pruning at initialisation.

6.3 Investigating Stability of a Saliency Metric

Even with a common pruning algorithm and a common goal, the results for pruning may vary from experiment run to experiment run. Most pruning algorithms include a stochastic element, it is impossible to find the exact same pruning rate when rerunning a pruning algorithm. However, in the pruning literature that we reviewed, pruning results are rarely presented with their variation.

In Chapters 3, 4, and 5, we quantified the variation in our experiments. We run each pruning experiment multiple times, and present error bars with 95% confidence intervals. We find that these errors bars were sometimes not negligible and they were not constant for all the saliency metrics. We can see these differences in Tables 3.11 and 3.12, where some saliency metrics have large error bars associated with them while others do not.

Reporting only the average result or the best result is not enough to decide whether a metric should be chosen or not. If stable results are required then the variation of the pruning algorithm should be considered. The degree of reliability to which pruning algorithms produce good results could be another metric by which pruning algorithms are ranked. Reliable pruning algorithms do not need to be run multiple times to approach their reported results thus reducing the overall cost of pruning.

Chapter 7

Conclusion

Channel pruning is an effective technique to reduce the size of CNNs. However, despite the extensive literature on channel pruning, there are a lot of questions to be answered to demystify pruning. In this thesis, we tried to provide answers to some of these questions. We propose different methods to improve the performance of existing metrics. Our research builds on top on existing research, and will most likely be compatible with any channel saliency metric.

- **Can we express saliency metrics in a standard form?**

In Chapter 3, we propose expressing saliency metrics, S, in a standard form, $S = \frac{1}{L} R \circ F(X)$ where X is the base input, F provides the pointwise metric, R, the reduction method, and L the scaling factor. By expressing saliency metrics in this standard form, not only are we facilitating their comparison with existing metrics, but we also bring to light some often forgotten aspects of the saliency metric, namely the reduction and scaling methods. The extensive saliency-based pruning literature reviewed for this thesis, contained a saliency metric that could be expressed in this standard form.

There is no standard way to empirically evaluate the performance of saliency metrics between research papers. Expressing saliency metrics into our proposed standard form facilitates finding commonalities and differences (if any) between them. For example, we found that metrics proposed as mask gradients (Lee et al. 2019), could also be expressed as Taylor expansions.

- **Given a standard form for saliency metrics, how do we construct good metrics?**

Using the summary of the evaluation of over 300 different saliency metrics in Chapter 3, we are able to provide guidance on the construction of saliency metrics for channel pruning. We found that there is no single saliency metric that provides the best results on all networks and all datasets. However, we did find commonalities between generally good metrics. We find that gradient-based metrics, like Taylor expansion-based metrics, are more likely to be provide good pruning rates. Furthermore, we find that using the feature map gradients gave slightly better results than using the weight gradients. We also find that Taylor expansion-based metrics that include both the first and second order terms lead to generally good results even with a cheap approximation for the latter. We find that scaling and reduction are important but are rarely discussed aspects of saliency metrics. Reduction methods that guarantee positive channel metrics lead to significantly better results. Our proposed scaling based on the number of weights transitively removed also improved pruning.

Our experiments were conducted using a subset of existing metrics. However, by expressing new saliency metrics into our standard form, they can also benefit from our findings about the different components of good saliency metrics.

- **Can multiple saliency metrics be used to find better decisions than a single metric?**

In Chapter 4, we propose one of the first solutions to use multiple saliency metrics within a single pruning algorithm. With our *myopic oracle*, we show how to combine the decisions of multiple metrics for each pruning iteration. We allow the pruning algorithm to choose pruning candidates from a pool of candidates voted by the different constituent metrics. At each iteration, the pruning algorithm may choose to use a different metric from the previous iteration or the same one. We combine the decisions of five popular saliency metrics that were initially built using different assumptions and different sets of information from the network. We show that the combined decisions of these multiple saliency metrics match or outperform the decisions of single metrics. Hence, the combination of saliency metrics can be used to find better decisions than a single metric.

Concurrently to our work, He et al. (2020) also proposed a method to use multiple saliency metrics within a pruning algorithm. However, apart from our work and the work from He et al. (2020) and Zhang et al. (2021), we are not aware of any other work exploiting the use of multiple saliency metrics for pruning. The use of multiple saliency metrics is a promising new technique that could be used to push the boundaries of pruning further.

- **How can channel saliency metrics incorporate information about other channels that may be removed?**

Channels that may be transitively removed as a consequence of pruning another channel are rarely explicitly discussed. This most commonly occurs in networks with splits and joins. However, the scarcity of explicit instructions on how to prune split and join connections hinders the reproduction of pruning experiments for these networks. Hence, before trying to improve channel saliency metrics for these networks, we had to explicitly describe how to prune channels that are part of split and join connections.

In Chapter 5, we propose *domino* saliency metrics to include structural information into the saliency metrics. We combine the saliency of channels that participate in common join and split nodes. The domino saliency metrics "see" the saliency of the other channels that can be removed. We find that the domino metric can significantly improve the base channel metric for networks with splits and joins. In particular, we find that the domino metrics making use of input channel saliency metrics lead to better pruning rates. The saliency of input channels provides valuable information for pruning split and join connections.

Final thoughts

The initial goal of this PhD was to improve saliency metrics for channel pruning. While we achieved what we set out to do, we also realised that our contributions were not limited to improving pruning rates. The findings of Chapters 3, 4, and

5 provide additional explainability to channel pruning. The standard form for saliency metrics proposed in Chapter 3 allows for easier reproducibility of saliency metrics. Our description of channel pruning in Chapter 5 also contributes to better reproducibility of pruning experiments as, without clear instructions, one is often left wondering how to handle split and join connections. Hence, the contributions of this PhD are also about demystifying channel pruning. We believe that pruning in general can be further improved by make pruning more reproducible and more explainable.

Appendices

Appendix A

Backpropagation of Layer-wise Hessian

A.1 Reminder of Backpropagation of Gradients

The weight gradients, $\frac{d\mathcal{L}}{dW}$, are found by backpropagating the feature map gradients, $\frac{d\mathcal{L}}{dA}$, through the network and obtained using the chain rule (Rumelhart et al. 1986).

For a given layer, l, the output feature map gradients $\frac{d\mathcal{L}}{d\,^{l}A}$ are known from the backpropagation of the $l+1^{th}$ layer. The output feature map, ^{l}A, the input feature map ^{l-1}A, and the weights ^{l}W are also known. During the backpropagation of the l^{th} layer, the aim is to find the weight gradients, $\frac{d\mathcal{L}}{d\,^{l}W}$, and the input feature map gradients, $\frac{d\mathcal{L}}{d\,^{l-1}A}$.

To simplify the written equations, we use $x = {}^{l-1}A[c,h,w]$, $a = {}^{l}A[m,i,j]$, and $w = {}^{l}W[m,c,u,v]$

A.1.1 Obtaining Weight Gradients

The multivariate chain rule for obtaining the weight gradients at the l^{th} layer is:

$$\frac{d\mathcal{L}}{dw} = \sum_{a \in {}^{l}A[m,i,j]} \frac{d\mathcal{L}}{da}\frac{da}{dw} \tag{A.1}$$

Using the chain rule in Equation A.1 and the equation of the convolution layer given in Equation 2.2, we can write down the equation to obtain the weight gradients of the l^{th} layer as:

$$\frac{d\mathcal{L}}{d\,{}^{l}W[m,c,u,v]} = \sum_{i=0}^{{}^{l}height-1} \sum_{j=0}^{{}^{l}width-1} \frac{d\mathcal{L}}{d\,{}^{l}A[m,i,j]} \frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l}W[m,c,u,v]} \tag{A.2}$$

The term $\frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l}W[m,c,u,v]}$ can be simplified into:

$$\frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l}W[m,c,u,v]} = {}^{l-1}A[c, i+u, j+v] \tag{A.3}$$

Using Equation A.2 and Equation A.3, we can rewrite the equation to obtain the weight gradients as gradients as:

$$\frac{d\mathcal{L}}{d\,{}^{l}W[m,c,u,v]} = \sum_{i=0}^{{}^{l}height-1} \sum_{j=0}^{{}^{l}width-1} \frac{d\mathcal{L}}{d\,{}^{l}A[m,h,w]}\,{}^{l-1}A[c, i+u, j+v] \tag{A.4}$$

Hence, for a convolution layer, a 2-dimensional convolution is required to obtain the gradient with respect to each individual weight.

A.1.2 Backpropagation of Feature Map Gradients

The multivariate chain rule for backpropagating the feature map gradients at the l^{th} layer is:

$$\frac{d\mathcal{L}}{dx} = \sum_{a \in {}^{l}A} \frac{d\mathcal{L}}{da}\frac{da}{dx} \tag{A.5}$$

Using the chain rule in Equation A.5 and the equation of the convolution layer given in Equation 2.2, we can write down the backpropagation of the l^{th} layer as:

$$\frac{d\mathcal{L}}{d\,{}^{l-1}A[c,h,w]} = \sum_{m=0}^{{}^{l}m-1} \sum_{i=0}^{{}^{l}height-1} \sum_{j=0}^{{}^{l}width-1} \frac{d\mathcal{L}}{d\,{}^{l}A[m,i,j]} \frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l-1}A[c,h,w]} \tag{A.6}$$

The term $\frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l-1}A[c,h,w]}$ can be simplified into:

$$\frac{d\,{}^{l}A[m,i,j]}{d\,{}^{l-1}A[c,h,w]} = {}^{l}W[m,c,h-i,w-j], \text{ if } 0 < h-i < {}^{l}k-1 \text{ and } 0 < w-j < {}^{l}k-1 \tag{A.7}$$

Using Equation A.6 and Equation A.7, we can simplify the backpropagation of feature map gradients through a convolution layer as:

$$\frac{d\mathcal{L}}{d\,{}^{l-1}A[c,h,w]} = \sum_{m=0}^{{}^{l}m-1} \sum_{i=h}^{h-{}^{l}k-1} \sum_{j=w}^{w-{}^{l}k-1} \frac{d\mathcal{L}}{d\,{}^{l}A[m,i,j]}\,{}^{l}W[m,c,h-i,w-j], \tag{A.8}$$

$$\text{if } h-i>0, \text{ and}$$
$$w-j>0,$$
$$\text{else } 0$$

If W^* is the weight matrix such that the 2-dimensional kernels of W are rotated by 180 degrees as given in Equation A.9, then Equation A.8 can be rewritten as Equation A.10.

$$W^*[m,c,u,v] = W[m,c,{}^{l}k-1-u,{}^{l}k-1-v], \text{ with } 0 \le m < {}^{l}m-1, \tag{A.9}$$
$$0 \le c < {}^{l}c-1,$$
$$0 \le u < {}^{l}k-1,$$
$$\text{and } 0 \le v < {}^{l}k-1$$

$$\frac{d\mathcal{L}}{d\,{}^{l-1}A[c,h,w]} = \sum_{m=0}^{{}^{l}m-1} \sum_{u=0}^{{}^{l}k-1} \sum_{v=0}^{{}^{l}k-1} \frac{d\mathcal{L}}{d\,{}^{l}A[m,h-{}^{l}k-1+u,w-{}^{l}k-1+v]}\,{}^{l}W^*[m,c,u,v], \tag{A.10}$$

$$\text{if } h-{}^{l}k-1+u>0$$
$$\text{and } w-{}^{l}k-1+v>0,$$
$$\text{else } 0$$

Equation A.10 shows that the backpropagation of feature map gradients through a convolution layer requires a convolution with the rotated weights.

A.2 Backpropagation of Diagonal Hessian Using Levenberg-Marquardt Approximation

LeCun et al. (1989) suggested a backpropagation of the diagonal Hessian using the Levenberg-Marquardt approximation to approximate the diagonal terms of the Hessian. In this section, we show how this backpropagation is implemented for convolution layers.

For the backpropagation of the second order derivatives, we assume that we have the same information available as the backpropagation of the gradients (feature maps, weights and their gradients) and the layer-wise Hessian of the $l + 1^{th}$ layer, $\frac{d^2\mathcal{L}}{d^{l+1}A^2}$. The aim is to find $\frac{d^2\mathcal{L}}{d^{l-1}A^2}$ and $\frac{d^2\mathcal{L}}{d^{l}W^2}$

We use the Faà di Bruno's formula for second order derivation to obtain the weight second order derivatives and propagate the feature map second order derivatives.

To simplify the written equations, we use $x = {}^{l-1}A[c, h, w]$, $a = {}^{l}A[m, i, j]$, and $w = {}^{l}W[m, c, u, v]$

A.2.1 Weight Hessian

We assume that the Hessian is diagonal. Hence, only the terms on the diagonal of the Hessian, $\frac{d^2\mathcal{L}}{dw^2}$, are required.

The Faà di Bruno formula generalises the chain rule for higher order derivatives. Faà di Bruno's formula for obtaining the second order weight derivatives in the multivariate case is given in Equation A.11.

$$\frac{d^2\mathcal{L}}{dw^2} = \sum_{a \in {}^{l}A} \left(\frac{d^2\mathcal{L}}{da^2} \left(\frac{da}{dw} \right)^2 + \frac{d\mathcal{L}}{dw} \frac{d^2 a}{dw^2} \right) \tag{A.11}$$

Using the Levenberg-Marquardt approximation $\frac{d\mathcal{L}}{dw}\frac{d^2 a}{dw^2}$ is neglected. For convolution layers, $\frac{d^2 a}{dw^2}$ is indeed equal to zero. The resulting approximation is shown in Equation A.12.

$$\frac{d^2\mathcal{L}}{dw^2} = \sum_{a \in {}^{l}A} \left(\frac{d^2\mathcal{L}}{da^2} \left(\frac{da}{dw} \right)^2 \right) \tag{A.12}$$

Using Equation A.12 and Equation 2.2, we obtain:

$$\frac{d^2\mathcal{L}}{d^lW[m,c,u,v]^2} = \sum_{i=0}^{^lheight-1}\sum_{j=0}^{^lwidth-1} \frac{d^2\mathcal{L}}{d^lA[m,i,j]^2}\left(\frac{d\,^lA[m,i,j]}{d\,^lW[m,c,u,v]}\right)^2 \tag{A.13}$$

$$= \sum_{i=0}^{^lheight-1}\sum_{j=0}^{^lwidth-1} \frac{d^2\mathcal{L}}{d^lA[m,h,w]^2}\,^{l-1}A[c,i+u,j+v]^2 \tag{A.14}$$

Hence, to obtain the Hessian with respect to the weights, 2-dimensional convolutions are required similar to the convolutions for obtaining the weight gradients.

A.2.2 Feature Map Hessian

The Hessian with respect to the feature maps is also assumed to be diagonal and the Levenberg-Marquardt approximation is used. The Faà di Bruno formula for second order derivatives is used to backpropagate the diagonal terms of the feature map Hessian. The backpropagation of the feature map Hessian is given in Equation A.15. After applying the Levenberg-Marquardt approximation, we obtain Equation A.16.

$$\frac{d^2\mathcal{L}}{dx^2} = \sum_{a\in^lA}\left(\frac{d^2\mathcal{L}}{da^2}\left(\frac{da}{dx}\right)^2 + \frac{d\mathcal{L}}{dx}\frac{d^2a}{dx^2}\right) \tag{A.15}$$

$$\frac{d^2\mathcal{L}}{dx^2} = \sum_{a\in^lA}\left(\frac{d^2\mathcal{L}}{da^2}\left(\frac{da}{dx}\right)^2\right) \tag{A.16}$$

Using Equation A.16, Equation 2.2 and Equation A.9, we obtain:

$$\frac{d^2\mathcal{L}}{d^{l-1}A[c,h,w]^2} = \sum_{m=0}^{^lm-1}\sum_{i=0}^{^lheight-1}\sum_{j=0}^{^lwidth-1} \frac{d^2\mathcal{L}}{d^lA[m,i,j]^2}\left(\frac{d\,^lA[m,i,j]}{d\,^{l-1}A[c,h,w]}\right)^2 \tag{A.17}$$

$$= \sum_{m=0}^{^lm-1}\sum_{u=0}^{^lk-1}\sum_{v=0}^{^lk-1} \frac{d^2\mathcal{L}}{d^lA[m,h-{^lk}-1+u,w-{^lk}-1+v]^2}\left(^lW^*[m,c,u,v]\right)^2, \tag{A.18}$$

if $h-{^lk}-1+u>0$ and $w-{^lk}-1+v>0$,

else 0

Hence, the backpropagation of the feature map Hessian using the method proposed by LeCun et al. (1989) can be implemented as a convolution with the rotated kernels squared.

We demonstrate that the backpropagation of second order derivatives used in LeCun et al. (1989), can be implemented as a convolution of similar structure as the backpropagation of gradients using the convolution layer given in Equation 2.2. The equation given does not include dilation, groups, and different padding strategies. However, even with convolution layers that include these, the backpropagation of second order derivatives results in a convolution of similar structure to the backpropagation of gradients.

A.3 Implementation

To implement the presented backpropagation of second order derivatives in Caffe (Jia et al. 2014), we extend the "blobs" in Caffe to store second order derivatives. The second order backpropagation of feature maps is implemented for all activation layers (in their test form). Since their second order derivation is trivial, we do not show them here. The second order backpropagation of the fully-connected layer can also easily be derived and is not presented here.

To implement the second order backpropagation for a convolution layer, we call a convolution with similar structure to the backpropagation of gradients but with different operands. To backpropagate the feature map gradient, instead of convolving the gradients of the previous layer with the weight tensor of rotated kernels, we convolve the second order derivatives with the weight tensor of rotated kernels squared. To obtain the weight second order derivatives, we use the same approach, i.e., we call a convolution with same structure as the one used to obtain the weight gradient and change the operands.

Bibliography

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. "Structured Pruning of Deep Convolutional Neural Networks". In: *ACM Journal on Emerging Technologies in Computing Systems* 13.3 (2017), 32:1–32:18.

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, and Klaus-Robert Müller. "On Pixel-wise Explanations for Non-linear Classifier Decisions by Layer-wise Relevance Propagation". In: *PLoS ONE* 10.7 (2015).

[3] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems (MLSys 2020), Austin, TX, USA, 2-4 March, 2020*. 2020.

[4] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. "High-Performance Large-Scale Image Recognition Without Normalization". In: *CoRR* abs/2102.06171 (2021). arXiv: `2102.06171`.

[5] Linhang Cai, Zhulin An, Chuanguang Yang, and Yongjun Xu. "Softer Pruning, Incremental Regularization". In: *Proc. International Conference on Pattern Recognition (ICPR 2020), Milan, Italy, 10-15 January, 2021*. 2021, pp. 224–230.

[6] Zhen Chen, Jianxin Lin, Sen Liu, Jun Xia, and Weiping Li. "Decouple and Stretch: A Boost to Channel Pruning". In: *Proc. IEEE International Performance Computing and Communications Conference (IPCCC 2018), Orlando, FL, USA, 17-19 November, 2018*. 2018, pp. 1–6.

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. "cuDNN: Efficient Primitives for Deep Learning". In: *CoRR* abs/1410.0759 (2014). arXiv: `1410.0759`.

[8] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. "A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets". In: *CoRR* abs/1707.08819 (2017). arXiv: `1707.08819`.

[9] *Counterclockwise: RAM capacity through the years*. URL: `https://www.gsmarena.com/counterclockwise_ram_capacity_through_the_years-news-30756.php` (Last Accessed on 11-August-2021).

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. "ImageNet: A large-scale hierarchical image database". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. 2009, pp. 248–255.

[11] Xiaohan Ding, Guiguang Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. "Global Sparse Momentum SGD for Pruning Very Deep Neural Networks". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8-14 December, 2019*. 2019, pp. 6379–6391.

[12] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman an Zhang, Eric Tzeng, and Trevor Darrell. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition". In: *Proc. International Conference on Machine Learning (ICML 2014), Beijing, China, 21-26 June, 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. 2014, pp. 647–655.

[13] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. "Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon". In: *Proceedings of Conference on Neural Information Processing Systems (NeurIPS 2017) Long Beach, CA, USA, 4-9 December, 2017*, 2017, pp. 4857–4867.

[14] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. "More is Less: A More Complicated Network with Less Inference Complexity". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), Honolulu, HI, USA, 21-26 July, 2017*. 2017, pp. 1895–1903.

[15] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *Proc. International Conference on*

Learning Representations (ICLR 2019), New Orleans, LA, USA, 6-9 May, 2019. 2019.

[16] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. "Stabilizing the Lottery Ticket Hypothesis". In: *CoRR* abs/1903.01611 (2020). arXiv: `1903.01611`.

[17] Akash Sunil Gaikwad and Mohamed El-Sharkawy. "Pruning the Convolution Neural Network (SqueezeNet) based on L2 Normalization of Activation Maps". In: *Proc. IEEE Computing and Communication Workshop and Conference (CCWC 2019), Las Vegas, NV, USA, 7-9 January, 2019*. 2019, pp. 392–396.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. 2016.

[19] Yiwen Guo, Anbang Yao, and Yurong Chen. "Dynamic Network Surgery for Efficient DNNs". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, 5-10 December, 2016*. 2016, pp. 1379–1387.

[20] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *Proc. International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, 2-4 May, 2016*. 2016.

[21] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. "DSD: Dense-Sparse-Dense Training for Deep Neural Networks". In: *Proc. International Conference on Learning Representations (ICLR 2017), Toulon, France, 24-26 April, 2017*. 2017.

[22] Song Han, Jeff Pool, John Tran, and William J. Dally. "Learning both Weights and Connections for Efficient Neural Network". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2015), Montreal, Quebec, Canada, 7-12 December, 2015*. 2015, pp. 1135–1143.

[23] Babak Hassibi and David G. Stork. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon". In: *Proceedings of Conference on Neural Information Processing Systems (NeurIPS 1992), Denver, Colorado, USA, 30 November - 3 December, 1992*. 1992, pp. 164–171.

[24] Babak Hassibi, David G. Stork, and Gregory J. Wolff. “Optimal Brain Surgeon: Extensions and Performance Comparison”. In: *Proceedings of Conference on Neural Information Processing Systems (NeurIPS 1993), Denver, Colorado, USA, 29 November - 2 December, 1993*. 1993, pp. 263–270.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, 27-30 June, 2016*. 2016, pp. 770–778.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proc. International Conference on Computer Vision (ICCV 2015), Santiago, Chile, 7-13 December, 2015*. 2015, pp. 1026–1034.

[27] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. “Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2020), Seattle, WA, USA, 13-19 June, 2020*. 2020, pp. 2006–2015.

[28] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. “Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks”. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13-19 July, 2018*. 2018, pp. 2234–2240.

[29] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. “Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16-20 June, 2019*. 2019, pp. 4340–4349.

[30] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. “AMC: AutoML for Model Compression and Acceleration on Mobile Devices”. In: *Proc. European Conference on Computer Vision (ECCV 2018), Munich, Germany, 8-14 September, 2018*. Vol. 11211. Lecture Notes in Computer Science. 2018, pp. 815–832.

[31] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22-29 October, 2017*. 2017, pp. 1398–1406.

[32] Joel Hestness, Newsha Ardalani, and Gregory F. Diamos. “Beyond Human-level Accuracy: Computational Challenges in Deep Learning”. In: *Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP 2019), Washington, DC, USA, 16-20 February, 2019*. 2019, pp. 1–14.

[33] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. “Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures”. In: *CoRR* abs/1607.03250 (2016). arXiv: 1607.03250.

[34] Kun Huang and Shenghua Gao. “Image saliency detection via multi-scale iterative CNN”. In: *The Visual Computer* 36.7 (2020), pp. 1355–1367.

[35] Zehao Huang and Naiyan Wang. “Data-Driven Sparse Structure Selection for Deep Neural Networks”. In: *Proc. European Conference on Computer Vision (ECCV 2018), Munich, Germany, 8-14 September, 2018*. Vol. 11220. Lecture Notes in Computer Science. 2018, pp. 317–334.

[36] Cheonghwan Hur and Sanggil Kang. “Entropy-based Pruning Method for Convolutional Neural Networks”. In: *Journal of Supercomputing* 75.6 (2019), pp. 2950–2963.

[37] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *Proc. ACM International Conference on Multimedia (MM 14), Orlando, FL, USA, 3-7 November, 2014*. 2014, pp. 675–678.

[38] Chunhui Jiang, Guiying Li, Chao Qian, and Ke Tang. “Efficient DNN Neuron Pruning by Minimizing Layer-Wise Nonlinear Reconstruction Error”. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13-19 July, 2018*. 2018, 2298–2304.

[39] Minsoo Kang and Bohyung Han. “Operation-Aware Soft Channel Pruning using Differentiable Masks”. In: *Proc. International Conference on Machine*

Learning (ICML 2020), virtual, 12-18 July, 2020. Vol. 119. Proceedings of Machine Learning Research. 2020, pp. 5122–5131.

[40] Devis Karaboga. *An Idea for Honey Bee Swarm for Numerical Optimization*. Kayseri, Turkey, 2005.

[41] Ehud D. Karnin. "A Simple Procedure for Pruning Back-propagation Trained Neural Networks". In: *IEEE Transactions on Neural Networks* 1.2 (1990), pp. 239–242.

[42] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[44] Andrew Lavin and Scott Gray. "Fast Algorithms for Convolutional Neural Networks". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, 27-30 June, 2016*. 2016, pp. 4013–4021.

[45] Duong H. Le and Binh-Son Hua. "Network Pruning That Matters: A Case Study on Retraining Variants". In: *Proc. International Conference on Learning Representations (ICLR 2021), virtual, Austria, 3-7 May, 2021*. 2021.

[46] Vadim Lebedev and Victor S. Lempitsky. "Fast ConvNets Using Group-Wise Brain Damage". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, 27-30 June, 2016*. 2016, pp. 2554–2564.

[47] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11. IEEE, 1998, pp. 2278–2324.

[48] Yann LeCun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 1989), Denver, Colorado, USA, 27-30 November, 1989*. 1989, pp. 598–605.

[49] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. “A Signal Propagation Perspective for Pruning Neural Networks at Initialization”. In: *Proc. International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26-30 April, 2020*. 2020.

[50] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. “SNIP: Single-shot Network Pruning based on Connection Sensitivity”. In: *Proc. International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6-9 May, 2019*. 2019.

[51] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. “Pruning Filters for Efficient ConvNets”. In: *Proc. International Conference on Learning Representations (ICLR 2017), Toulon, France, 24-26 April, 2017*. 2017.

[52] Sheng R. Li, Jongsoo Park, and Ping Tak Peter Tang. “Enabling Sparse Winograd Convolution by Native Pruning”. In: *CoRR* abs/1702.08597 (2017). arXiv: `1702.08597`.

[53] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. “Pruning and Quantization for Deep Neural Network Acceleration: A Survey”. In: *CoRR* abs/2101.09671 (2021). arXiv: `2101.09671`.

[54] Yongsheng Liang, Wei Liu, Shuangyan Yi, Huo-Xiang Yang, and Zhenyu He. “Filter pruning-based two-step feature map reconstruction”. In: *Signal, Image and Video Processing* 15.7 (2021), pp. 1555–1563.

[55] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: *CoRR* abs/1312.4400 (2013). arXiv: `1312.4400`.

[56] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. “Channel Pruning via Automatic Structure Search”. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2020), Yokohama, Japan, 7-15 January, 2021*. 2020, pp. 673–679.

[57] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. “Accelerating Convolutional Networks via Global & Dynamic Filter Pruning”. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13-19 July, 2018*. 2018, pp. 2425–2432.

[58] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. "Progressive Neural Architecture Search". In: *CoRR* abs/1712.00559 (2017). arXiv: `1712.00559`.

[59] Congcong Liu and Huaming Wu. "Channel Pruning based on Mean Gradient for Accelerating Convolutional Neural Networks". In: *Signal Processing* 156 (2019), pp. 84–91.

[60] Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. "Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey". In: *CoRR* abs/2005.04275 (2020). arXiv: `2005.04275`.

[61] Jing Liu, Bohan Zhuang, Zhuangwei Zhuang, Yong Guo, Junzhou Huang, Jinhui Zhu, and Mingkui Tan. "Discrimination-aware Network Pruning for Deep Model Compression". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1.

[62] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K.H. So. "Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers". In: *Proc. International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26-30 April, 2020*. 2020.

[63] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. "Efficient Sparse-Winograd Convolutional Neural Networks". In: *Proc. International Conference on Learning Representations (ICLR 2018), Vancouver, BC, Canada, 30 April - 3 May, 2018*. 2018.

[64] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. "MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning". In: *Proc. International Conference on Computer Vision (ICCV 2019), Korea (South), 27 October - 2 November, 2019*. 2019, pp. 3295–3304.

[65] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. "Frequency-Domain Dynamic Pruning for Convolutional Neural Networks". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2018), Montreal, Quebec, Canada, 3-8 December, 2018*. 2018, pp. 1051–1061.

[66] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. "Learning Efficient Convolutional Networks through Network Slimming". In: *Proc. International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22-29 October, 2017*. 2017, pp. 2755–2763.

[67] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. "Rethinking the Value of Network Pruning". In: *Proc. International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6-9 May, 2019*. 2019.

[68] Jian-Hao Luo and Jianxin Wu. "AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference". In: *Pattern Recognition* 107 (2020), p. 107461. ISSN: 0031-3203.

[69] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression". In: *Proc. International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22-29 October, 2017*. 2017, pp. 5068–5076.

[70] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. "ThiNet: Pruning CNN Filters for a Thinner Net". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.10 (2019), pp. 2525–2538.

[71] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. "Exploring the Limits of Weakly Supervised Pretraining". In: *Proc. European Conference on Computer Vision (ECCV 2018), Munich, Germany, 8-14 September, 2018*. Vol. 11206. Lecture Notes in Computer Science. 2018, pp. 185–201.

[72] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. "Exploring the Granularity of Sparsity in Convolutional Neural Networks". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), Honolulu, HI, USA, 21-26 July, 2017*. 2017, pp. 1927–1934.

[73] Gaurav Menghani. "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better". In: *CoRR* abs/2106.08962 (2021). arXiv: `2106.08962`.

[74] Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, and Balaraman Ravindran. “Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks”. In: *Proc. 2018 IEEE Winter Conference on Applications of Computer Vision (WACV 2018), Lake Tahoe, NV, USA, 12-15 March, 2018*. 2018, pp. 848–857.

[75] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. “Importance Estimation for Neural Network Pruning”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16-20 June, 2019*. 2019, pp. 11264–11272.

[76] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. “Pruning Convolutional Neural Networks for Resource Efficient Inference”. In: *Proc. International Conference on Learning Representations (ICLR 2017), Toulon, France, 24-26 April, 2017*. 2017.

[77] Michael Mozer and Paul Smolensky. “Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment”. In: *Proceedings of Conference on Neural Information Processing Systems (NeurIPS), Denver, Colorado, USA, 1988]*. 1988, pp. 107–115.

[78] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. “ASAP: Architecture Search, Anneal and Prune”. In: *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS 2020), Virtual, 26-28 August, 2020*. Vol. 108. Proceedings of Machine Learning Research. 2020, pp. 493–503.

[79] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. “Collaborative Channel Pruning for Deep Networks”. In: *Proc. International Conference on Machine Learning (ICML 2019), Long Beach, California, USA, 9-15 June, 2019*. Vol. 97. Proceedings of Machine Learning Research. 2019, pp. 5113–5122.

[80] Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V. Le. “Meta Pseudo Labels”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2021), virtual, 19-25 June, 2021*. 2021, pp. 11557–11568.

[81] Adam Polyak and Lior Wolf. “Channel-Level Acceleration of Deep Face Representations”. In: *IEEE Access* 3 (2015), pp. 2163–2175.

[82] Ramchalam Kinattinkara Ramakrishnan, Eyyub Sari, and Vahid Partovi Nia. "Differentiable Mask for Pruning Convolutional and Recurrent Networks". In: *Proc. Conference on Computer and Robot Vision (CRV 2020), Ottawa, ON, Canada, 13-15 May, 2020*. 2020, pp. 222–229.

[83] Russell Reed. "Pruning algorithms - A Survey". In: *IEEE Transactions on Neural Networks* 4.5 (1993), pp. 740–747.

[84] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. "Scaling Vision with Sparse Mixture of Experts". In: *CoRR* abs/2106.05974 (2021). arXiv: `2106.05974`.

[85] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[86] Marcel Simon, Erik Rodner, and Joachim Denzler. "ImageNet pre-trained models with batch normalization". In: *CoRR* abs/1612.01452 (2016). arXiv: `1612.01452`.

[87] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: `1409.1556`.

[88] Suraj Srinivas and R. Venkatesh Babu. "Data-free Parameter Pruning for Deep Neural Networks". In: *Proc. British Machine Vision Conference (BMVC 2015), Swansea, UK, 7-10 September, 2015*. 2015, pp. 31.1–31.12.

[89] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), Boston, MA, USA, 7-12 June, 2015*. 2015, pp. 1–9.

[90] Hidenori Tanaka, Daniel Kunin, Daniel L. Yamins, and Surya Ganguli. "Pruning neural networks without any data by iteratively conserving synaptic flow". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2020), virtual, 6-12 December, 2020*. 2020.

[91] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. “SCOP: Scientific Control for Reliable Neural Network Pruning”. In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2020), virtual, 6-12 December, 2020*. 2020.

[92] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. “Faster Gaze Prediction with Dense Networks and Fisher Pruning”. In: *CoRR* abs/1801.05787 (2018). arXiv: `1801.05787`.

[93] Richard Tomsett, Dan Harborne, Supriyo Chakraborty, Prudhvi Gurram, and Alun D. Preece. “Sanity Checks for Saliency Metrics”. In: *Proc. AAAI Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7-12 February, 2020*. 2020, pp. 6021–6029.

[94] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. “Fixing the train-test resolution discrepancy”. In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8-14 December, 2019*. 2019, pp. 8250–8260.

[95] Jack Turner, José Cano, Valentin Radu, Elliot J. Crowley, Michael F. P. O’Boyle, and Amos J. Storkey. “Characterising Across-Stack Optimisations for Deep Convolutional Neural Networks”. In: *Proc. IEEE International Symposium on Workload Characterization (IISWC 2018), Raleigh, NC, USA, 30 September 30 - 2 October, 2018*. 2018, pp. 101–110.

[96] Aravind Vasudevan, Andrew Anderson, and David Gregg. “Parallel Multi Channel convolution using General Matrix Multiplication”. In: *Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2017), Seattle, WA, USA, 10-12 July, 2017*. 2017, pp. 19–24.

[97] Chaoqi Wang, Guodong Zhang, and Roger Grosse. “Picking Winning Tickets Before Training by Preserving Gradient Flow”. In: *Proc. International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26-30 April, 2020*. 2020.

[98] Huan Wang, Qiming Zhang, Yuehai Wang, and Haoji Hu. “Structured Probabilistic Pruning for Convolutional Neural Network Acceleration”. In: *Proc.*

British Machine Vision Conference (BMVC 2018), Newcastle, UK, 3-6 September, 2018. 2018, p. 149.

[99] Zhenyu Wang, Fu Li, Guangming Shi, Xuemei Xie, and Fangyu Wang. "Network Pruning using Sparse Learning and Genetic Algorithm". In: *Neurocomputing* 404 (2020), pp. 247–256.

[100] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Learning Structured Sparsity in Deep Neural Networks". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, 5-10 December, 2016*. 2016, pp. 2074–2082.

[101] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated Residual Transformations for Deep Neural Networks". In: *CoRR* abs/1611.05431 (2016). arXiv: `1611.05431`.

[102] Sheng Xu, Anran Huang, Lei Chen, and Baochang Zhang. "Convolutional Neural Network Pruning: A Survey". In: *Proc. Chinese Control Conference (CCC 2020), Shenyang, China, 27-29 July, 2020*. 2020, pp. 7458–7463.

[103] Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. "Balanced Sparsity for Efficient DNN Inference on GPU". In: *CoRR* abs/1811.00206 (2018). arXiv: `1811.00206`.

[104] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. "Pruning by explaining: A novel criterion for deep neural network pruning". In: *Pattern Recognition* 115 (2021), p. 107899. ISSN: 0031-3203.

[105] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. "Gate Decorator: Global Filter Pruning Method for Accelerating Deep Convolutional Neural Networks". In: *Proc. Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8-14 December, 2019*. 2019, pp. 2130–2141.

[106] Jiecao Yu, Andrew Lukefahr, David J. Palframan, Ganesh S. Dasika, Reetuparna Das, and Scott A. Mahlke. "Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism". In: *Proc. International Symposium on Com-*

puter Architecture (ISCA 2017), Toronto, ON, Canada, 24-28 June, 2017. 2017, pp. 548–560.

[107] Jiecao Yu, Jongsoo Park, and Maxim Naumov. "Spatial-Winograd Pruning Enabling Sparse Winograd Convolution". In: *CoRR* abs/1901.02132 (2019). arXiv: `1901.02132`.

[108] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. "NISP: Pruning Networks Using Neuron Importance Score Propagation". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018), Salt Lake City, UT, USA, 18-22 June, 2018*. 2018, pp. 9194–9203.

[109] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. "Scaling Vision Transformers". In: *CoRR* abs/2106.04560 (2021). arXiv: `2106.04560`.

[110] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander J. Smola. "ResNeSt: Split-Attention Networks". In: *CoRR* abs/2004.08955 (2020). arXiv: `2004.08955`.

[111] Yidan Zhang, Youheng Zhen, Zhenan He, and Gray G. Yen. "Improvement of Efficient in Evolutionary Pruning". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2021), Shenzhen, China, 18-22 July, 2021*. 2021.

[112] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning". In: *Proc. International Conference on Learning Representations (ICLR 2017), Toulon, France, 24-26 April, 2017*. 2017.