Inducing Sparsity in Deep Neural Networks through Unstructured Pruning for Lower Computational Footprint

Camille Ballas (MSc.)

A dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (PhD.)

to the

Dublin City University – School of Computing

Supervisors:

Dr. Suzanne Little and Prof. Noel E. O'Connor

August 2022

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Camille Ballas

Id No.: 17213942

Date: 24/08/2022

Contents

1 Introduction **1**
- 1.1 Deep learning era . . . 2
- 1.2 Towards more efficient Deep Learning . . . 4
- 1.3 Deep Learning compression . . . 7
- 1.4 Hypothesis & Research Questions . . . 9
  - 1.4.1 Hypothesis . . . 10
  - 1.4.2 RQ1: Pruning Framework . . . 11
  - 1.4.3 RQ2: Integrity of unstructured pruning heuristics . . . 11
  - 1.4.4 RQ3: The impact of a sparse neural network on fairness . . . 11
- 1.5 Thesis Outline . . . 12

2 Background: Deep Learning fundamentals **15**
- 2.1 Feed forward networks and basic concepts . . . 16
  - 2.1.1 Multi Layers Perceptrons . . . 17
  - 2.1.2 Training cycle . . . 18
  - 2.1.3 Running inference . . . 22
- 2.2 Deep Learning for image processing . . . 23
  - 2.2.1 Convolutional Neural Networks . . . 24
  - 2.2.2 Over-parameterisation . . . 31
- 2.3 Scope of this thesis . . . 37
  - 2.3.1 Model evolution dynamics . . . 37
  - 2.3.2 Datasets . . . 41
  - 2.3.3 Model architectures . . . 42
- 2.4 Chapter Summary . . . 43
- 2.5 Chapter conclusion . . . 44

3 Internet-of-Things and Applied AI **47**
- 3.1 Use-case: Crowd monitoring for Smart City . . . 48
  - 3.1.1 Croke Park: Smart Stadium for Smarter living . . . 49
  - 3.1.2 Performance of video-processing at the edge . . . 52
  - 3.1.3 Deploying video-processing in real-world environment . . . 55
- 3.2 Reducing computational cost . . . 56
  - 3.2.1 Measuring computational cost . . . 57
  - 3.2.2 Deep learning model compression . . . 60
  - 3.2.3 On Hardware limitations . . . 62
- 3.3 Chapter Summary . . . 68

4 The state of sparsity and network pruning **71**
4.1 Pruning methodology . . . 72
4.1.1 Pruning masks . . . 73
4.1.2 Pruning criterion and saliency . . . 75
4.1.3 Structured versus Unstructured pruning . . . 75
4.1.4 Pruning ratio and scope . . . 77
4.1.5 Reducing computation . . . 78
4.2 A short history of unstructured pruning . . . 80
4.2.1 A way to reduce over-parameterisation . . . 80
4.2.2 Deploying AI in productions . . . 81
4.2.3 Better understanding deep learning training . . . 81
4.2.4 Complexity of pruning research . . . 83
4.3 Pruning Frameworks . . . 84
4.3.1 One-shot . . . 84
4.3.2 Iteratively . . . 85
4.3.3 Multi-stage . . . 88
4.4 Chapter Summary . . . 89

5 Pruning Metrics **91**
5.1 Pruning criteria families . . . 93
5.1.1 Magnitude pruning . . . 93
5.1.2 Loss-modelling . . . 94
5.1.3 Gradient based . . . 98
5.2 Locality assumption . . . 101
5.2.1 Multi-stage pruning . . . 102
5.2.2 Constraining step-size . . . 103
5.2.3 Others considerations . . . 104
5.3 Pruning Integrity . . . 104
5.3.1 Performance Metrics . . . 106
5.3.2 Methodology . . . 108
5.4 Experimental results . . . 110
5.4.1 Loss-preservation criteria . . . 110
5.4.2 Gradient-flow preservation . . . 115
5.4.3 Scaling to ImageNet . . . 124
5.4.4 Improving GraSP criteria . . . 124
5.5 Chapter Summary & Discussion . . . 125
5.5.1 Problem Statement . . . 125
5.5.2 Discussion & Conclusion . . . 126

6 Beyond accuracy: evaluating pruning performance **129**
6.1 On the impact of parameterisation . . . 130
6.1.1 Methodology: Wide verses Deep network . . . 131
6.1.2 Experimental results & observations . . . 134
6.1.3 Discussion and Limitations . . . 144
6.2 On Fairness and Robustness . . . 148
6.2.1 Methodology: Comparing fairness of pruning criteria . . . 149
6.2.2 Experimental results . . . 151
6.2.3 Discussion and limitations . . . 156
6.3 Chapter Summary . . . 158

7 Conclusions **161**
7.1 Hypothesis and Research Questions . . . 162
7.1.1 RQ1: Pruning Framework . . . 162
7.1.2 RQ2: Integrity of unstructured pruning heuristics . . . 163
7.1.3 RQ3: The impact of a sparse neural network on fairness . . . 165
7.2 Contribution . . . 166
7.3 Limitations . . . 168
7.4 The future of unstructured pruning . . . 169
7.4.1 Practical Impact . . . 169
7.4.2 Research implications . . . 171
7.4.3 Societal Impact . . . 172

Appendix **173**

A **175**
A.1 Smart Stadium experimental setups . . . 175
A.1.1 Hardware . . . 175
A.1.2 Software . . . 175
A.2 Deep Learning Orchestration (DeepLO) . . . 176
A.2.1 DeepLo testbed hardware specifications . . . 176

B **179**
B.1 Details on the Experimental Setup (section 5.4 . . . 179
B.1.1 Setups . . . 179
B.1.2 Experiments . . . 180
B.1.3 Pruning Framework . . . 181

C **183**
C.1 Supplementary materials for Wide and Deep networks experiments (Section 6.1) . . . 183
C.1.1 Training Hyper-parameters . . . 183
C.1.2 Models accuracy . . . 185
C.1.3 Fine-tuning . . . 185
C.1.4 Layer collapsing . . . 185
C.2 Supplementary materials for fairness in pruned networks (section 6.2) 190
C.2.1 Training Hyper-parameters . . . 190

List of publications **210**

List of Figures

1.1 Computation complexity, measured in floating point operations (Flop) per second per day for different deep learning models over the years. 2012 corresponds to the year GPUs started being used to train deep learning models. We can observe that from this point there is an exponential growth in model complexity. Reproduced from Amodei et al. (2018) 3

1.2 Number of parameters for different model architectures, for computer vision applications (left) and natural language processing (right). From Menghani (2021), Figure 1. 5

1.3 Number of paper published around sparsity over the years. Sourced from Hoefler et al. (2021), Fig. 1. 10

1.4 High-level organisation of the thesis. 14

2.1 XOR Bolean operation represented in a 2D space (**left**) for a binary input x_1 (squares) and x_2 (triangle). The associated MLP graph (**right**) pictures the three different perceptron units h_1, h_2 and y and their parameters values. The 1 circle represent the bias for each hidden units. Image reproduced from the *Probabilitic Machine Learning* book (Murphy, 2022) 17

2.2 illustration of a **neuron unit** where x_i represent the input data, w_i and b are respectively the weights and biases parameters, and ϕ the activation function. 19

2.3 Different type of activation functions ϕ. The most common activation used in practice is the ReLU – $ReLU(a) = \mathbf{max}(a, 0)$ for hidden layers because it is better at propagating the signal through deep networks. Image issued from the *Probabilistic Machine Learning* book (Murphy, 2022). 20

2.4 Fully connected layer where x_i are the input and s_i activation. From Goodfellow et al. (2016) *Deep Learning* book. 25

2.5 Illustration of convolution operations: (left) numerical operation, and (right) spatial filtering. Source: Murphy (2022) and (Robles, 2018) . . 26

2.6 Reproduced from *Probabilitic Machine Learning* book (Murphy, 2022). 28

2.7 VGG16 architecture, where 16 represent the number of layers (convolutional and fully connected). {`filter-size`} `conv,` {`number-filter`} refers to convolutional layers, and `fc` {`number-neurons`} corresponds to fully connected layers. After each group of conv layers a max-pooling operation is applied, with `sizeXXX` indicating the input size for the layer group. Reproduced from 29

2.8 Illustration of different CNN architectures: ResNet-34 (top), plain-34 (middle) and VGG-19 (bottom). Skip connections allows the deep neural network to be trained beyond 34 layers, dashed residual connections represent `CONV-1x1` from succession of two layers that differ in size. Source He et al. (2015). 30

2.9 Illustration of a residual blocks with a skip connection: (left) *post-activation* ResNet, and (right) *pre-activation* as in PreAct ResNet Source: He et al. (2016b) 30

2.10 Bias-variance trade-off. Reproduce from Belkin et al. (2019) 32

2.11 Illustration of different type of *drop-like* regularisation for a MLP network. While removing neurons (b) automatically removes all associated weights, removing weights does not necessarily shut down the associated neuron. However, if all the incoming connections (or weights) are removed as pictured in (c) middle-line, removing all the connections is equivalent to removing a neuron. 36

2.12 Illustration of the minimisation problem that the gradient descent algorithm is trying to solve, where $f(x)$ represents the loss function. Reproduced from the Deep Learning book Goodfellow et al. (2016). . 38

2.13 Momentum update. Taken from Sebastian Ruder's blog article *Optimizing Gradient Descent* (https://ruder.io/optimizing-gradient-descent). 40

2.14 Sample images of three different datasets: (a) MNIST, (b) CIFAR10. 41

3.1 Croke Park smart stadium sensors map. 50

3.2 Fog computing paradigm 51

3.3 Impact on reducing the image input size on the speed. The bigger the data input size, the higher the number of compute operation will be and thus the algorithm will require more compute power. 55

3.4 Compute performance evolution over the years of GPUs and CPUs measured in GFLOPS. Nowadays there exist even more powerful GPUs with the deep learning standard Nvidia V100 reaching 7 teraFLOPS in double-precision mode. It is a no match compared to CPUs arithmetical compute power. Reproduced from Karl (2013). . . 58

3.5 DeepLO testbed pipeline. Workloads are deployed on different hardware through docker containers. Different metrics are collected and stored into a database to run deeper analytics off-line. 63

3.6 Frame per second (fps) ratio for different workloads – levels of compression (colour) – for different model architectures (columns) on the various hardware tested (rows). The behaviour of different workloads is not consistent across different hardware, especially for ResNet-20 workloads. We can observe good fps rates on i5nuc and i7nuc, but on fpr1 and fpr2, the performances are worst than VGG19 while being less computationally demanding (see Table 3.3). up2 hardware is not powerful enough to handle any workload, while ResNet-164 is too complex to be handled efficiently by any hardware. Note that the sudden drop offs observed on the ResNet20 is due to system issues. . 64

3.7 CPU utilisation (in %) for different compressed workloads for VGG19 (top row) and Resnet20 (bottom row) on different hardware (columns). We can notice that VGG19 is fairly consistent on all devices whiles Resnet20 fluctuates especially on frp1. up2 does not have the compute power required to handle any of the different workload properly. 67

4.1 Illustration of pruning methodology where $\frac{1}{3}$ of the parameters are removed per layer. $\boldsymbol{\theta}_i$ represent the weight matrice for parameters $\boldsymbol{\theta}$ at layer i. First, the saliency metric is computed, parameters are then ranked by order of importance – in this example the higher the better – and the ones falling below the threshold are removed. A pruning mask is derived from the previous ranking with 0 values for parameters to be discarded and 1 for parameters remaining in the model. An element-wise product between the dense network and the pruning mask is applied to obtain the sparse network. 74

4.3 Illustration of compressed-sparse-row (CSR) storage requirements to store sparse representation efficiently. 79

4.4 *One-shot* pruning. A straightforward way of removing parameters where the model is pruned and train only once. 84

4.5 *Iterative* pruning. The cycle of pruning-retraining is performed n times to iteratively remove small chunks of parameters at a time. By performing the pruning cycle multiple time, the pruning heuristic adjust itself and we are able to better preserve the network architecture. 86

4.6 *Lottery Ticket* framework. Similar to iterative pruning presented in Figure 4.5, pruning is perform multiple times at the exception that weights are set back to their values at initialisation prior to fine-tuning, training the sparse network from scratch each pruning cycle. . 87

4.7 *Multi-stage* pruning. Similar to iterative pruning, the pruning is performed multiple times but we do not retrain the sparse model in-between each pruning steps. This lower the computations while removing parameters little at a time. 88

5.1 $\|\Delta\boldsymbol{\theta}\|_2$ for different level of sparsity and multi-stage schedule strategies, oneshot (orange square), linear (blue dots), exponential (green triangles). 103

5.2 Pruning accuracy for different pruning methods applied at initialisation. We can observe that all pruning criteria lie in close range. For information a 10^2 corresponds to 99% pruning ratio. Reproduced from Tanaka et al. (2020). 105

5.3 $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ for different number of pruning stages π, as a function of λ, the step size constraint strength, using either (left) LM, (middle) QM or (right) OBD criteria. MP, which is invariant to λ and to the number of pruning stages, is displayed in dashed black. The curves are the mean and the error bars the standard deviation over 5 random seeds. OBD with $\pi = 1$ and $\lambda = 0$ diverged for all of the 5 seeds. Increasing the number of pruning stages drastically reduces $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. A $\lambda > 0$ can also help improving performances. 111

5.4 Same as Figure 5.3, but displaying the validation error gap before fine-tuning. With proper number of pruning stages and step size regularization, LM and QM can produce pruned networks that are drastically better than the ones pruned using MP. 114

5.5 Gap of validation error after fine-tuning as a function of $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. We can observe a slight correlation on the MLP network architectures, however this does not translate to larger scale architectures VGG11 and PreActResnet18. 116

5.6 Left: Using the same hyper-parameters for fine-tuning as the ones of the original training. Right: Performing hyper-parameters optimisation for the fine-tuning. 117

5.7 $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ for different number of pruning stages π, as a function of λ. As for the loss-modelling criteria, augmenting the number of π help to better preserve the gradient flow before/after pruning. λ regularisation however requires a more careful tuning as the magnitude of the saliencies issued from the gradient-based criteria can be high. . 118

5.8 Similar to Figure 5.7, but displaying the validation error gap before fine-tuning instead of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. 120

5.9 Gap of validation error after fine-tuning as a function of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. 121

5.10 Gap of validation error after fine-tuning as a function of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. Zoomed in view of Figure 5.9 (center) for gradient-based criteria on VGG11. When we ignore points corresponding to pruned-networks that were unable to be re-trained (validation error

$0.9), we do not observe any significant correlations.$ 122

5.11 Same as Figure 5.3 and Figure 5.5, for the ResNet50 on ImageNet, with a sparsity of 70 %. Increasing the number of pruning stages and constraining the step size reduce $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. However, the best-loss preserving criteria, which maximize the validation accuracy right after pruning, do not produce better networks after fine-tuning. They perform similarly if their validation accuracy after pruning is $> 20\%$. Criteria that outperform MP right after pruning do not achieve better performance after fine-tuning. 123

5.12 Same as Figure 5.11, but with 90 % sparsity. Increasing the number of pruning stages and constraining the step size reduces $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. However, the best-loss preserving criteria, which maximize the validation accuracy right after pruning, do not produce better networks after fine-tuning. 123

5.13 Same at Figure 5.7, $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ for different number of pruning stages π, as a function of λ for $\|\text{GraSP}\|$ pruning criteria. We observe that removing the lowest magnitude of GraSP, in addition to regularisation λ and multi stage pruning π lead to better performance than the original GraSP heuristic. We are able to prevent the gradient norm to being too large. 125

6.1 Different model architectures considered for studying the impact of parameterisation over pruning criteria integrity, MLP (left) and Convnet (right). Every layer is followed by ReLU activations, and a Softmax at the end for the classifier. 133

6.2 Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a MLP with 2 layers with 512 neurons on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored 135

6.3 Scatter plot displaying the $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ *before/after pruning* as a function of the best validation error obtained after fine-tuning (where α indicates the scaling factor). We can observe that there is a range of networks for $\alpha = 4$ (MLP with 8 layers) where the sparse model failed to retrain accurately leading to high error accuracy for low change in gradient flow. 136

6.4 Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a Convnet trained on CIFAR-10 with 2 layers with 64 filters on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored. 137

6.5 Training curves for different baseline model (dense architecture) for MLP (*top*) and Convnet on CIFAR-10 (*middle*) and Convnet on MNIST (*bottom*). For each, we display the baseline for different scaling factors α. 138

6.6 Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a Convnet trained on MNIST with 2 layers with 64 filters on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored. 140

6.7 Fine-tuning curves (classification error) for different model architectures scaled on the depth. (See Figure C.1.3.2 in Appendix C for model architectures scaled on the width.) 142

6.8 Scatter plot of the average divergence between the training and validation curves verses the correlation score between best preserving the original network function ($\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ and $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$) and end-accuracy of the pruned model. 143

6.9 Classification error (plain line) as a function of the sparsity ratio for a Convnet trained on MNIST. The dense model accuracy is reported in grey. Dotted line display the average gap between the validation and training set error. 146

6.10 Ratio of parameters remaining per layer (x axis) for a VGG16 architecture when pruned before training (top), at the middle of training (middle) or after training (bottom). Magnitude pruning is applied in this example. Different colors correspond to different ratio of pruning, with blue being a small pruning ratio and red high pruning ratio. . . 147

6.11 Per class prediction deviation for different pruning criteria when pruning is applied at the end of training. Classes are represented along the x-axis. 153

6.12 Number of classes with significant mean-shift class accuracy between different pruning criteria and the baseline – dense model, where 10 is the maximum number of classes. It can be observed that all the pruning criteria diverged significantly from the baseline indicating that pruning does not preserve the original model prediction distribution. Moreover, criteria preserving the original network function (LM, $\|GraSP\|$ and Synflow) observed similar behaviour with only a little shift in class prediction. 154

6.13 Similar to Figure 6.12 but for pruning at initialisation. Per class prediction deviation for different pruning criteria when pruning is applied before training. 155

6.14 Comparison of number of classes with significant mean-shift class accuracy between pruning criteria. Similar to Figure 6.12 but for pruning at initialisation. 156

6.15 Example PIEs images obtained when pruning after training. 157

A.1 Software architecture for crowd monitoring 176

A.2 List of the different hardware metrics monitored to assess the cost of running deep learning workloads on constrained devices. 177

C.1 Fine-tuning curves (loss) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance. 186

C.2 Fine-tuning curves (classification error) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance. 187

C.3 Fine-tuning curves (loss) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance. 188

C.4 Ratio of parameters remaining per layer (x axis) for a VGG19 architecture when pruned before training (top), at the middle of training (middle) or after training (bottom). Magnitude pruning is applied in this example. Different colors correspond to different ratio of pruning, with blue being a small pruning ratio and red high pruning ratio. . . 189

List of Tables

2.1 Summary of the complexity of fully-connected and convolutional layers, where I is the input, O the output, K_w and K_h are respectively the kernel width and height, C the number of channels, H and W the height and width of the tensor (input or output), p and s correspond to the *padding* and *striding* of the convolution operation. 27

3.1 Crowd monitoring algorithms tested to assess compute power abilities at the edge. The complexity is measured as the number of compute operations required to process one data point and is arbitrarily expressed to compare the different algorithms with one another. 53

3.2 Performance comparison reported as Frames-per-second (Fps) and Seconds to process frame (Spf) for both hardware devices tested. The **I5-3210M CPU** is between twice to four time more efficient at running crown monitoring algorithms than the **Atom E3825 CPU**. 54

3.3 Comparison of the properties of different compressed workload for VGG19 and Resnet20 models. Resnet20 is much lighter model to run compare to VGG19. Compression can greatly reduce the number of parameters and FLOPs to help deploy deep learning workload on constrained devices. 66

5.1 Summary of the best $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ across values of λ for different networks and pruning criteria, with $\pi = 140$. QM achieves better loss-preservation than other criteria. OBD performs worse than QM, since we violate its convergence assumption when pruning in several stages. 112

5.2 Best validation error gap **before/after pruning** for different networks and pruning criteria. 113

5.3 Summary of the best $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ across values of λ for different networks and pruning criteria. The best performing criteria varies a lot depending on the network architecture. GraSP and SynFlow are good at preserving the norm of the gradients. 119

5.4 Best validation error gap **before/after pruning** for different networks and loss-modelling pruning criteria. 119

5.5 Best validation error gap **before/after fine-tuning** the pruned networks (lower is better), for different pruning criteria, across values of λ and π. GraSP and LM are among the best performing criteria, but all the criteria validation error gap lie in a close neighborhood after fine-tuning. 122

6.1 Summary of different pruned model statistics. The 2nd column report the model accuracy after fine-tuning, the 3rd the number of class that shifted significantly from the baseline (dense) model, and finally the 4th column corresponds to the number of data points that observed a change in classifications between the dense and sparse networks outcome – note that CIFAR10 contains 10k datapoints. As a reference, the accuracy for the dense model is **90.90% ± 0.31** 152

6.2 Summary of different pruned model statistics. The 2nd column report the model accuracy after fine-tuning, the 3rd the number of class that shifted significantly from the baseline (dense) model, and finally the number of data points that observed a change in classifications – note that CIFAR10 contains 10k datapoints. As a reference, the accuracy for the dense model is **90.90% ± 0.31** 155

A.1 CPU specification for the Smart Stadium experiments. 175

A.2 Specifications of the different hardware in the DeepLO testbed. Note that frp2 is a more recent version of the frp1 aquired later in the project. 178

C.1 Summary of the best validation error across different *width scaling* for different pruning criteria. The number following each criterion name indicates the number of local iteration (π) performed during pruning. We observe that all networks are reaching good performances indicating no problems encountered during pruning or fine-tuning. . . 184

C.2 Summary of the best validation error across different *width scaling* for different pruning criteria. The number following each criterion name indicates the number of local iteration (π) performed during pruning. 184

Inducing Sparsity in Deep Neural Networks through Unstructured Pruning for Lower Computational Footprint

Camille Ballas

Abstract

Deep learning has revolutionised the way we deal with media analytics, opening up and improving many fields such as machine language translation, autonomous driver assistant systems, smart cities and medical imaging to only cite a few. But to handle complex decision making, neural networks are getting bigger and bigger resulting in heavy compute loads. This has significant implications for universal accessibility of the technology with high costs, the potential environmental impact of increasing energy consumption and the inability to use the models on low-power devices. A simple way to cut down the size of a neural network is to remove parameters that are not useful to the model prediction. In unstructured pruning, the goal is to remove parameters (ie. set them to 0) based on some importance heuristic while maintaining good prediction accuracy, resulting in a high-performing network with a smaller computational footprint. Many pruning methods seek to find the optimal capacity for which the network is the most compute efficient while reaching better generalisation. The action of inducing sparsity – setting zero-weights – in a neural network greatly contributes to reducing over-parametrisation, lowering the cost for running inference, but also leveraging complexity at training time. Moreover, it can help us better understand what parts of the network account the most for learning, to design more efficient architectures and training procedures. This thesis assesses the integrity of unstructured pruning criteria. After presenting a use-case application for the deployment of an AI application in a real-world setting, this thesis demonstrates that unstructured pruning criteria are ill-defined and not adapted to large scale networks due to the over-parametrisation regime during training, resulting in sparse networks lacking regularisation. Furthermore, beyond solely looking at the performance accuracy, the fairness of different unstructured pruning networks is evaluated highlighting the need to rethink how we design unstructured pruning.

Acknowledgements

Doing a PhD has always been a big dream of mine, and now I can proudly say that I did it! This journey was not without difficulties and I would not have been able to succeed without the incredible support of the people around me.

First I would like to acknowledge Insight Centre for Data Analytics and Science Foundation Ireland (SFI) under Grant Number 12/RC/2289_P2 and 16/SP/3804 that have made this thesis possible. To my supervisors, Suzanne and Noel, thank you for always supported my projects and guiding me throughout my 4 years of PhD. Especially to Suzanne, you always made sure I was not drowning in my work and that my comfort was the number one priority during the Covid pandemic that impacted the final years of this PhD. I also want to thank the jury for their time and consideration in reading my Thesis and for their useful feedback.

But the ones that made sure I kept a life outside my PhD are my friends and family, you are the ones that gave me the strength and motivation to pursue this PhD until the end. To my friends here in Dublin and back in France, Rodrigo, Ricardo, Elise, Aude, thank you for making sure I took the time to relax and enjoy life. To my partner, Loris, thank you for always being there for me when I needed you. Finally, to my family, you have always been me supporting me, and without your incredible support, I would not have been able to make it.

I would also like to thank my colleagues, Eric, Dian, Kevin, Feiyan, César and Eva for making most of the days in the lab fun and showing me the road to being a good researcher. To Eva in particular, you have been an amazing mentor and friend above all, thank you!

Finally, I would like to say a huge thanks to my Discord fellow PhD friends that have made writing a thesis under lockdown much more bearable and even enjoyable. Catherine, Isabelle and Manon, you have been the real sunshine of those months and I was happy to work because it meant listening to silly music with you.

But if I had to dedicate this thesis to one person it would be my brother, I cannot thank you enough for being the amazing brother and mentor you have been during my PhD, I am truly grateful for it. If I am the researcher I am today it is thanks to your guidance and support during the hardest time. Thank you!

Chapter 1

Introduction

Over the last decade advances in Artificial Intelligence, led by the progress in deep learning research, have transformed the field of media analytics opening up new possibilities. The rise of deep learning has lead to a paradigm shift from traditional feature engineering to a data-driven approach. Deep learning research has now reached a certain maturity and is being used for large-scale, real-world applications.

However, deep learning models are overly parameterised. They possess far more parameters than training examples, which while providing good knowledge representation, might incite a model to memorise the data instead of learning to solve the task for which it has been trained. As a result, the computational power needed to train state-of-the-art models often requires multiple high-performing GPU devices, increasing not only their carbon footprint, but also expanding the disparities between practitioners who can afford compute cost and those who cannot. Moreover, over-parameterised models also have greater storage requirements, which hinders their deployment in real-world applications as the computational power and capacity of devices may not be sufficient to support the latest neural networks. In summary, the computational complexity of deep learning algorithms constitutes a major bottleneck.

To tackle these issues, researchers have explored ways to reduce the number of parameters in deep learning models. Model compression methods have been developed to leverage the computational complexity of deep learning models and create

more efficient architectures by either condensing the original model into a smaller one (Hinton et al., 2015; Han et al., 2015a; Plummer et al., 2021; Howard et al., 2019; Liu et al., 2017), or inducing sparsity by replacing non important parameters by zero-values (LeCun et al., 1989; Zhu & Gupta, 2017; Lee et al., 2019; Wang et al., 2020; Tanaka et al., 2020). Sparsity methods in particular are promising due to their versatility and ability to greatly reduce the number of parameters required right from training time.

1.1 Deep learning era

The traditional computer approach to solve complex tasks, features would be engineered to extract useful knowledge from the data. For instance, to compare two images we cannot just compare their RGB values pixel-wise. A small change in lighting or camera angle, can result in two very different RGB grids pixel-wise. To tackle this issue, researchers have spent years engineering descriptors such as scale invariant features transform (SIFT) to capture useful knowledge from images such as object position or orientation, beyond their specific numerical value. This is equivalent to data-aggregation with features in other non-vision applications.

The appeal behind deep learning has been motivated by the capacity of deep neural networks to learn complex features to perform powerful data analytics on images, videos, texts, without having to design features extracted by hand. Neural networks are able to learn useful representations across a wide range of applications from computer vision to natural language processing. Many applications nowadays rely heavily upon deep learning algorithms. From language translation and generation (Chan et al.; Brown et al., 2020), to image generation (Radford et al., 2021), medical applications (Shen et al., 2017), autonomous driving (Grigorescu et al., 2020), physical system modelling and analysis (Rasp et al., 2018; Hezaveh et al., 2017), there is a wide variety of domains that benefit from deep neural networks.

The rise of deep learning has been driven by an increase in data-collection and an improvement in compute power with specialised compute-oriented hardware (GPUs,

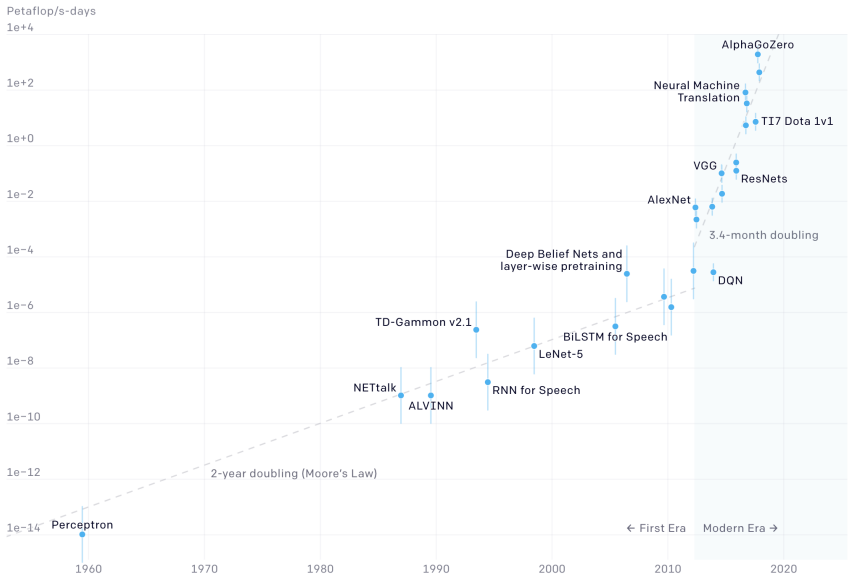


Figure 1.1: Computation complexity, measured in floating point operations (Flop) per second per day for different deep learning models over the years. 2012 corresponds to the year GPUs started being used to train deep learning models. We can observe that from this point there is an exponential growth in model complexity. Reproduced from Amodei et al. (2018)

TPUs). In the early years of deep learning, neural networks were limited by the compute power available – models were exclusively trained and run on CPUs. As a result, they were limited in capacity and only composed of a few layers with less than a million internal parameters, limiting their analytical power. For these reasons they were not widely used outside academia. In 2012, the ImageNet (Deng et al., 2009; Krizhevsky et al., 2012) revolution created new opportunities when both data and compute power became available simultaneously. Researchers Alex Kizhevsky and his colleagues created a deep model architecture with 60M parameters named Alexnet (Krizhevsky et al., 2012), capable of reaching close to human-level performance on the ImageNet image classification task. To train their AI they took advantage of the compute power of GPUs to learn from a huge quantity of training data, opening up the door to more affordable and efficient training of very deep networks. Nowadays, training of deep learning models is exclusively done on GPUs or more specialised hardware subsequently developed (e.g. TPUs).

Following their work, a wide variety of new deep learning models were created

(VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2015), Inception (Szegedy et al., 2016), etc.). Slowly deep learning expanded from computer vision application to other areas like natural language processing, text and image generation, and so on. Over the past 10 years, researchers have created deeper and bigger network architectures to continue pushing the boundaries. As a result the algorithm complexity, including the workload required to train deep learning models, have observed an exponential growth (see Figure 1.1). The current state-of-the-art model in natural language processing OpenAI GPT-3 (Brown et al., 2020), used by companies like Microsoft in their code generation tools GitHub Copilot (Langston, 2021), contains 175 billion parameters, a 174 billions increase over AlexNet in under 10 years.

To tackle more ambitious and real world problems, the number of parameters in deep network architectures has kept increasing alongside the amount of training data required to achieve top performances. This constitutes a major drawback to deploying deep learning applications in production. The cost of training those giant deep learning models has also widened the gap between practitioners due to compute power accessibility, with the big tech companies Google, Microsoft or Facebook monopolising the research panorama. An additional important concern is the sustainability of deep learning if compute requirements continue to increase. In an era where the climate crisis is at the uppermost centre of attention, the carbon footprint (energy consumption and hardware requirement) of deep learning cannot continue to be ignored. This raises questions about the necessity for over-parameterisation in modern deep learning models, when (Denil et al., 2013) showed that sometimes only 5% of parameters matters for prediction. What role does over-parameterisation play in model performance? Can we successfully learn without it?

1.2 Towards more efficient Deep Learning

Over-parameterisation hinders the compute requirement of deep neural networks as it increases the compute capacity required to train and run deep learning models. But it also involves the use of extensive regularisation techniques to prevent the

model from over-fitting (memorising) the training data. So why do we need over-parameterised model in the first place? Empirical studies (Advani et al., 2020; Nakkiran et al., 2021) suggest that over-parameterisation helps facilitate the training of deep algorithms on complex tasks but it isn't without shortcomings.

An algorithm's efficiency can be measured in terms of compute resources required to execute the program, the objective being to complete the task within a minimal compute time. The compute requirement of a deep learning model depends on three main factors: (1) the number of parameters in the model, (2) the amount of floating point operations (FLOPs), and (3) the data input size. The first two depend on the model architecture while the last one is derived from the training data. To understand how to build more efficient deep learning models, we first need to understand the role each of those factors play on the compute load.

Number of parameters The number of parameters influences the size of the model, and increases the requirements to store and load the model into memory. Over the years, the number of parameters within deep learning models has drastically increased from 100M parameters for early computer-vision models up to 175B parameters for latest NLP models (see Figure 1.2). Note that parameters are usually encoded as float32 data, thus each single parameter occupies 8 octets.

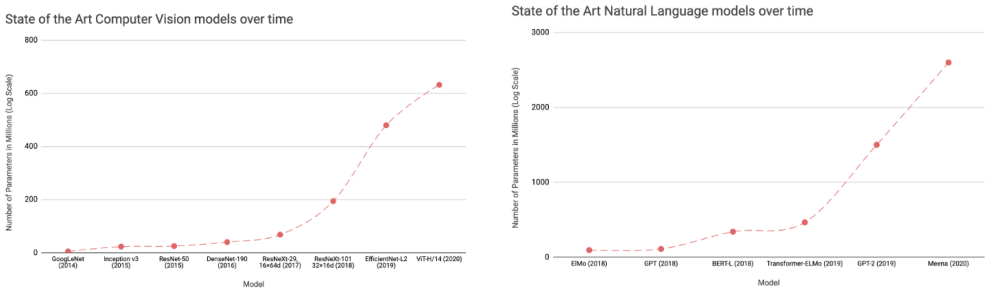


Figure 1.2: Number of parameters for different model architectures, for computer vision applications (left) and natural language processing (right). From Menghani (2021), Figure 1.

It is also worth noting that a large number of parameters doesn't always imply a bigger compute load as larger models can have lesser computational complexity due to floating point operations.

Floating point operations (Flops) The number of floating point operations corresponds to the number of simple mathematical operations (multiplication, subtraction, addition, or division) required to execute the algorithm and affects the computational workload (GPU or CPU usage). As presented in Figure 1.1, the amount of compute load has grown exponentially over the past 10 years.

Data input size Finally, the input size has an impact on both the memory and compute workload of the algorithm. Large input data may imply more parameters to process the input, and thus more floating point operations. In this thesis, the main interest is deep learning model efficiency, therefore the impact of the input on compute resources is not in the scope of this study and will be not be considered.

There are two scenarios for which one might want to improve efficiency: (1) for faster training and better generalisation performance to unseen data, or (2) to reduce the computational requirement of inference and run on low power hardware devices.

Reducing the number of parameters at training time is essential to ensure a sustainable future for deep learning research. Current deep network architectures are far from being efficient parameter-wise (Bianco et al., 2018; Hoefler et al., 2021), but their high parameterisation enables easier optimisation and highly expressible models. Therefore finding more efficient architectures while preserving their high potential is of utmost importance. Over-parameterisation is still not well understood and finding more efficient deep network architectures is an ongoing topic (Cortes et al., 2017; Cai et al., 2019; Stamoulis et al., 2020). But recent research suggests that there exists within a deep network a sub-network that can be trained to solve the task in isolation (Frankle & Carbin, 2018). How to find such sub-network? This remains an active research question.

When deploying deep learning applications in production, we are mostly interested in *inferences*. That is, we only care about the computational performance for model prediction. In practice, to manage the heavy workload it is common to rely on a cloud-centric approach to take advantage of unlimited and scalable com-

pute power. However, this has some inconveniences as it required reliance on a third party agent. Two majors concerns are the network bandwidth limitation and privacy threats. For certain use cases such as autonomous driver assistance, high responsiveness is required, and thus latency induced by cloud services is a major issue. For other use-cases like crowd monitoring or medical imaging, we deal with extremely sensitive data, thus it is not desirable to allow data to transit through a third party.

For both cases, the application can directly be deployed on-device, at the edge, to increase the response time and reinforce privacy. But even with specialised hardware like neural compute chips (Dinelli et al., 2019), the model may still be too large and too complex for the device's computational capabilities. Therefore, it is of interest to reduce the complexity of deep learning models at the source, for more efficient and compact models.

1.3 Deep Learning compression

Deep learning compression can benefit from both reducing computational cost at inference, and decreasing the number of parameters at training time. When compressing a deep neural network we look to reduce the number of parameters, and thus the computational complexity, while maintaining the performance of the original dense network (prediction accuracy). Many compression methods have been developed over the years: (1) pruning, (2) factorisation or quantisation, (3) distillation or hardware oriented architectures.

In *pruning* Han et al. (2015a) the aim is to remove non important parameters from the model (ie, setting them to zero) based on some importance measures while maintaining good prediction accuracy, resulting in a high-performing network with a smaller computational footprint. To do so, we compute a *saliency* or *importance* score for each individual parameter. Parameters can then be ranked in order of importance, and the ones with the smallest saliency scores are pruned (removed) from

the network, while the ones with the largest scores are kept unchanged. The resulting pruned model has the property of being *sparse*, analogous to the mathematical definition of sparse matrices.

Factorisation, or *quantisation* tries to reduce the dimensions of the parameter space by aggregating, or clustering, parameters together (Zhao et al., 2019a; Han et al., 2015a; Zhang et al., 2018). Some methods are even able to further accelerate inferences by shifting the model parameter space to a binary or ternary space (Bethge et al., 2019; He et al., 2019). Factorisation has proven to be very efficient to reduce the cost of running deep learning applications in production as it directly reduces the number of mathematical operations alongside reducing memory storage. However, such methods struggle to leverage the computational complexity at training time.

To further improve efficiency of deep networks, one can directly employ smaller *hardware oriented model architectures* specifically designed to be efficient on low-power devices (Howard et al., 2019). This can be done through Knowledge Distillation (Hinton et al., 2015) or Neural Architecture Search (Stamoulis et al., 2020). Such models have the advantage of being directly optimised for constrained devices but they require lots of expertise to develop.

Amongst all compression methods mentioned above, inducing sparsity through network *pruning* in a deep neural network is the most versatile method. Due to its simplicity and flexibility, it can be easily applied before, during or after training making pruning a popular approach to compress deep learning models. It has proven to be successful to reducing the size of a model up to 90% without impairing the performance of the network (Han et al., 2015a; Liu et al., 2017), suggesting that there exists more efficient and better performing small architectures. For that reason, pruning methods are very promising. But removing parameters from the original network is not done without harm. Finding the right parameters to remove can be laborious and computationally expensive. Removing parameters almost always results in a loss in accuracy requiring to re-train the compressed model to recover the original model performance, and despite being of similar accuracy, the pruned

model often encompass greater bias in the model prediction producing less fair models (Hooker et al., 2021).

In order to provide more efficient and sustainable AI, it is important to better understand the role of parameterisation and how to reduce it. Sparsity, and more specially pruning, can help identify what part of the network accounts the most for the learning, what properties of the network lead to better performance. This can promote designing better, cost efficient, neural network architectures with as little parameters as possible. Note that pruning can be applied in a *structured* way, where parameters are removed in close groups, or in an *unstructured* way, where parameters are removed individually. The latter is a more flexible approach as it has fewer constraints over any enforced sparsity pattern, and thus can better help us understand inherent dynamics in deep network training.

1.4 Hypothesis & Research Questions

Neural networks are getting bigger, requiring increased computational resources not only for training, but also for inference. This has significant implications for the universal accessibility of the technology with high compute costs, potential environmental impact and difficulty in deploying deep learning application on low-power devices.

Unstructured pruning compression aims at removing parameters from the model based on some *importance measures* that determine their usefulness at predicting the outcome decision, while maintaining good prediction accuracy. Pruning has attracted lots of attention recently due to its simplicity and high potential. A wide variety of pruning heuristics and algorithms have been developed over the span of the last couple of years (see Figure 1.3). But pruning is also an efficient way to induce and explore the role of sparsity in a network to help understand how to reach the optimal capacity – ie, reduced memory footprint with high generalisation property – from an over-parametrised deep learning model.

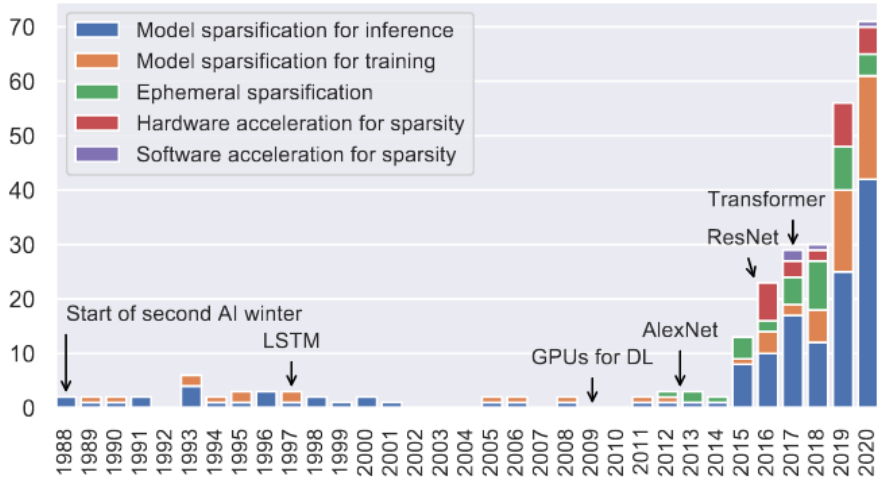


Figure 1.3: Number of paper published around sparsity over the years. Sourced from Hoefler et al. (2021), Fig. 1.

1.4.1 Hypothesis

This thesis analyses the benefits and limitations of unstructured pruning methods to create sparse neural networks. We concentrate our focus on computer vision applications as it is where most attention and recent discoveries in pruning have happened in the last decade due to the maturity of the field. The overall hypothesis of this work can be expressed as follows:

Deep learning models are over-parametrised. Only a small subset of parameters account for most of the learning and prediction, and by restricting the training or inference to this sub-network, we can drastically reduce computational complexity without losing accuracy. There exists an optimal way to identify parameters importance in a deep learning model such that accuracy and fairness of the original model can be preserved.

To explore this hypothesis we review different aspects of the pruning methodology. We study three main aspects of network pruning: (1) the framework by which we induce sparsity, (2) the importance measures used to determine which parameters to discard, and finally, (3) the impact of sparse neural networks on the network performance. By exploring those components, this intend to provide guidance in

regards to inducing deep learning sparsity through unstructured pruning.

1.4.2 RQ1: Pruning Framework

What pruning framework is the most efficient to reduce the size of a network?

This research question examines the limitations of unstructured pruning and explores how different pruning strategies impact the performance of the compressed model. For that, Chapter 4 provides an extensive review of the pruning literature, highlighting major discoveries and frameworks that have been developed over the past decade. We investigate the limitations of pruning toward building more efficient pruning frameworks.

1.4.3 RQ2: Integrity of unstructured pruning heuristics

How good are existing pruning criteria at producing sparse models?

Measuring the importance of a parameter is the most important factor for inducing sparsity, as it directly determines the outcome of the pruned network. If the wrong parameters are removed, then the pruned network will be useless. In this research question, we explore the core foundation of pruning criteria and investigate which characteristics of different pruning heuristics lead to better performing pruned models. In Chapter 5 an empirical study of six major unstructured pruning criteria is conducted. We demonstrate that despite their wide adoption amongst the community, pruning criteria fail at consistently producing good sparse models. Chapter 6.1 further explores the role architecture plays on pruning performance and offers some insights as of why pruning heuristics fail.

1.4.4 RQ3: The impact of a sparse neural network on fairness

Beyond solely looking at the prediction accuracy, does the fairness of the pruned model differ following different pruning strategies?

Over-parameterisation is helpful for training deep neural networks, but only a small subset of parameters are accountable for the learning and can reach similar performance. This research question explores the cost of removing parameters from the original model, and its implication for performance beyond prediction accuracy. Experiments to investigate implicit biases induced by different pruning methods is conducted in Chapter 6.2.

1.5 Thesis Outline

For understanding the rest of this thesis, Chapter 2 will review the foundation for training and running inference with a deep learning model. The different components (optimiser, regularisations, architectures) that contribute to the learning will be presented. A section will be dedicated to over-parameterisation and measuring the computational complexity of deep neural networks.

Following to this, a case study for deploying AI applications in a real-world setting will be presented in Chapter 3 to further illustrate how the compute requirement impair deep learning progress. Practical tools to leverage compute complexity from a hardware perspective will also be presented.

In Chapter 4 the foundation of sparsity and network pruning will be presented. Different different frameworks developed over the years will be reviewed and compare to one another. Through this chapter we will explore and answer RQ1.

Once the core foundations of pruning have been introduced, Chapter 5 is dedicated to the investigation of pruning criteria design. Experimental results are presented in order to assess the integrity of unstructured pruning metrics for compressing deep neural networks. We address RQ2 and review different types of pruning criteria motives to study whether they help improve the performance of the pruned model.

The next chapter, Chapter 6, explores RQ3 and take a look at the impact of pruning compression and sparsity beyond prediction accuracy. We will investigate potential bias induced by unstructured sparsity, and study to what extent removing

parameters impacts the model representation.

Finally we will conclude this thesis by offering a guidance with a set of rules to help practitioners apply unstructured pruning sparsity in deep learning models in Chapter 7.

The reader can refer to Figure 1.4 for guidance regarding the organisation of this thesis.

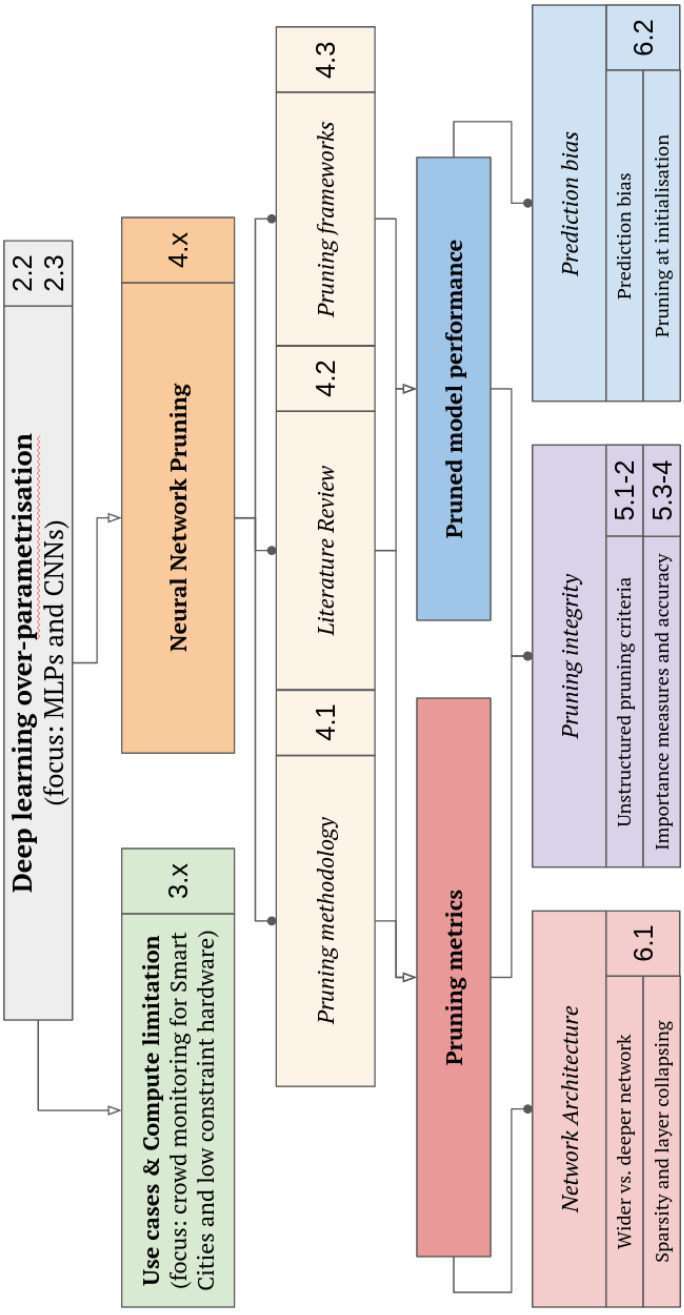


Figure 1.4: High-level organisation of the thesis.

Chapter 2

Background: Deep Learning fundamentals

Artificial intelligence has been around for decades. Since the very first moment programmable computers were created, scientists questioned the potential for such machines to become intelligent (Lovelace, 1842). What is intelligence? For an algorithm to be intelligent means it is able to take actions in order to reach a given goal (Poole & Mackworth, 2017).

There exists many levels of intelligence. Computers can easily solve some complex tasks that are difficult for a human, given a set of rules and a constraint environment. In 1997, IBM was able to create an algorithm named *Deep Blue* competent enough to defeat the world chess champion Garry Kasparov (Hsu, 2002). But chess contains a finite number of actions and solutions, for other tasks where rules cannot be formulated explicitly, computers struggle. For a human recognising objects or judging similarity in an image is straightforward, but this has proven to be a surprisingly challenging problem for computers.

Deep learning has revolutionised the world of artificial intelligence thanks to its ability to handle complex unstructured data such as text and images. Instead of giving the algorithm a set of rules, we let the model learn from experience and define its own representation to solve the problem. To tackle computer vision and natural

language problems, deep neural networks contain a large number of parameters to maximise knowledge extraction.

The core idea explored in this thesis of reducing the size of a deep learning model to achieve the best out of a small subset of parameters, inherently relies on the basic foundations of deep learning. While this can be achieved on any type of deep network architecture and for diverse applications, this thesis focuses on computer vision tasks relying on Fully-Connected and Convolutional Neural Networks that will be presented below in this chapter. I chose to limit the scope of my study to image classification application due to its simplicity and wide adoption amongst the pruning community, making it a good candidate for empirical theoretical research. Indeed, unlike natural language models, or generative adversarial networks, Fully-Connected and Convolutional Neural Networks used to solve image classification tasks do not rely on latent space or feedback connections which simplify the training and analysis of the deep network behaviour.

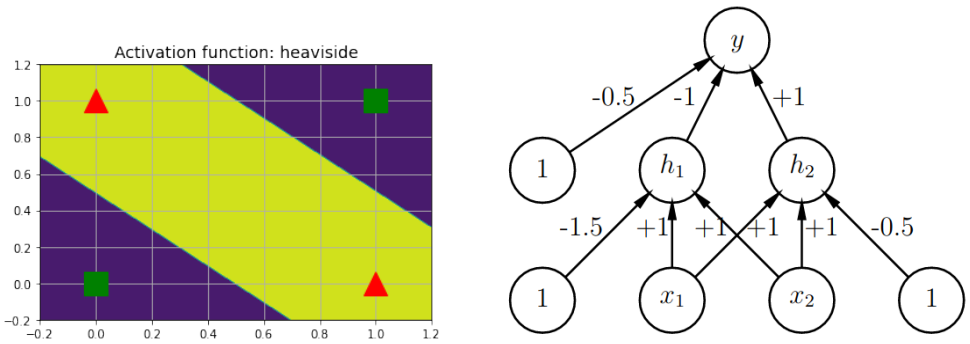
Thus, this chapter introduces the technical background essential for understanding the research presented in this thesis. Section 2.1.2 introduces the basic concepts for training and running inference with a deep neural network. In section 2.2 the reader is familiarised with the concept of parameters and neural network architectures applied to computer vision problems. Finally section 2.3 lists the different models and datasets that will be considered for investigation in this thesis.

2.1 Feed forward networks and basic concepts

Deep learning is a broad family of machine learning algorithms that rely on differentiable functions organised in a *directed acyclic graph* that map the input to the output. In contrast to a majority of machine learning algorithms – e.g., linear regressions, generalised linear models – deep learning models do not assume a linear relationship between the input and output.

2.1.1 Multi Layers Perceptrons

Feed forward networks, also called *Multi-Layer perceptron* (MLP) represent the core mechanism of deep neural networks. A popular case study to illustrate the principle of perceptron units is the XOR dilemma (Minsky & Papert, 1969). To create a model that learns to compute the Boolean function exclusive OR, a linear mapping between the input and output is not possible, as illustrated in Figure 2.1. It is not possible to draw a single line that can successfully separate the triangles from the squares (Figure 2.1, left). However it is possible to add multiple perceptron units to produce intermediate features representation to help computing the XOR output (Figure 2.1, right).



(a) (b)

Figure 2.1: XOR Bolean operation represented in a 2D space (**left**) for a binary input x_1 (squares) and x_2 (triangle). The associated MLP graph (**right**) pictures the three different perceptron units h_1, h_2 and y and their parameters values. The 1 circle represent the bias for each hidden units. Image reproduced from the *Probabilitic Machine Learning* book (Murphy, 2022)

Given two inputs x_1 and x_2, the XOR function can be decomposed as follow $y = f(x_1, x_2) = \overline{(x_1 \text{ AND } x_2)}$ AND $(x_1$ OR $x_2)$. Both the AND and OR operation can be approximated using a linear thresholding function $f(x, \boldsymbol{\theta}) = \mathbb{I}(\mathbf{w}^T x + \mathbf{b} \geq 0)$ computed through perceptron units:

$$H(x) = f(x; \boldsymbol{\theta}) = \mathbf{W}^T x + \mathbf{b} \geq 0 \tag{2.1}$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ are the parameters of the model. Replacing the formulas in

the context of the XOR problem, this leads to $h_1 = (x_1 \text{ AND } x_2) = H(\mathbf{W}_1{}^T\mathbf{x} + b_1)$, $h_2 = (x_1 \text{ OR } x_2) = H(\mathbf{W}_2{}^T\mathbf{x} + b_2)$, and finally $y = (h_1 \text{ AND } h_2) = H(\mathbf{W}_3{}^T\mathbf{a} + b_3)$, where h_1, h_2 and y are all peceptron units, $\mathbf{x} = [x_1, x_2]$ and $\mathbf{a} = [h_1, h_2]$.

In order for the output unit y to compute the XOR $\{0, 1\}$ output, weights $\mathbf{W}$ and biases $\mathbf{b}$ need to be carefully tuned such that h_1, h_2 and y produce the desired output defined above. Ideal parameters are reported in Figure 2.1 (left). This example illustrates the core foundation of deep neural networks: multiple perceptron or compute units organised in multiple layers, such that useful representations, or features, can be extracted from the input data to produce the desired output. Note that perceptron units belonging to the intermediate layers are also called *hidden units* (h_i units in the previous XOR example). Perceptron units can be generalised to:

$$H(x) = f(x; \boldsymbol{\theta}) = \mathbf{W}^T\phi(x) + \mathbf{b} \tag{2.2}$$

where ϕ represent the *activation function* for feature transformation. Deep neural networks have great potential to encompass complex and high dimensional decision making. However, it is not feasible to define every single parameter by hand, especially for neural networks where models easily contain thousands of parameters. For that reason, the neural network is trained to learn the best values for parameters from experience (data).

2.1.2 Training cycle

To understand how to reduce the size of a deep neural network, it is important to understand how different parameters influence the training and prediction of the model and the role that they play. Training a neural network is similar to training any linear machine learning model, in the sense that it follows a *gradient-based* optimisation procedure. Parameters are optimised to reduce a *cost function* (or *loss* function) $\mathcal{L}$ that evaluates the model performance based on some training data. In other words, the loss function measures how wrong the model prediction is compared

with its true target. If deep learning follows a supervised learning procedure, where the network is taught to solve a task based on training examples, there exists other training procedures – semi-supervised (Oliver et al., 2018) and unsupervised (Vos et al., 2017) – that extend over the training procedure that will be described in this section but consideration of these training methods is outside the scope of this thesis.

Training a deep neural network relies on gradient-based learning. It can be decomposed into three steps: (1) the *forward pass* during which, given an input, the model computes the output prediction, (2) the *backward pass* that estimates how far the model prediction is from the true ground-truth and computes new values for the parameters that would better solve the tasks, and, (3) the *update pass* that updates the parameters following the optimisation steps.

2.1.2.1 Forward pass

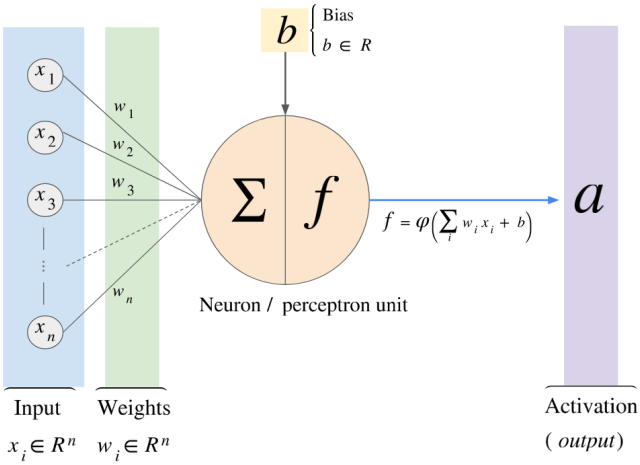


Figure 2.2: illustration of a **neuron unit** where x_i represent the input data, w_i and b are respectively the weights and biases parameters, and ϕ the activation function.

MLPs, and deep neural networks in general, are a succession of hidden layers containing multiple neural units that can be summarised as:

$$\boldsymbol{z}_l = \phi_l(\boldsymbol{a}_l) \tag{2.3}$$

$$\boldsymbol{a}_l = \boldsymbol{b}_l + \boldsymbol{W}_l \boldsymbol{z}_{l-1} \tag{2.4}$$

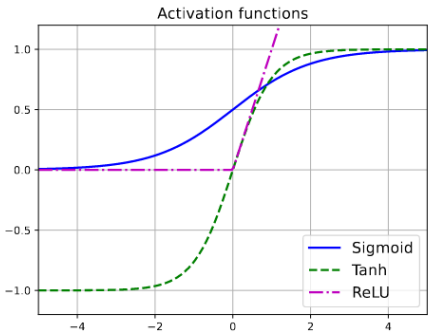


Figure 2.3: Different type of activation functions ϕ. The most common activation used in practice is the ReLU – $ReLU(a) = \mathbf{max}(a, 0)$ for hidden layers because it is better at propagating the signal through deep networks. Image issued from the *Probabilistic Machine Learning* book (Murphy, 2022).

where $\boldsymbol{\theta}_l = [\boldsymbol{W}_l; \boldsymbol{b}_l]$ are the parameters of given layers, l, $\boldsymbol{z_l}$ the hidden unit, $\boldsymbol{a}$ the *pre-activation* that represents the input signal ingested by the neurons, and ϕ the activation function.

The **forward pass** simply consists of processing the input data $\boldsymbol{X}$ through the neural network to compute the output y. The input signal flowing through the network is altered by the different neuron units, extracting new representations of the input data. The choice of the activation function ϕ for the hidden and output layers is of central importance as it defines the behaviour of the deep learning model. If ϕ is a linear transform $\mathbb{I}(\boldsymbol{W}x + \boldsymbol{b})$, then the mapping between the input $\boldsymbol{X}$ and output $\boldsymbol{y}$ would be linear too. However, the choice of activation function does not impact the structure of the parameters $\boldsymbol{\theta}$ in itself as the weights $\boldsymbol{W}$ and bias $\boldsymbol{b}$ are not altered by the function ϕ (as illustrated in Figure 2.2 and equation 2.4). The most common activation function in practice for the hidden layer are $ReLU$ activations, and *softmax* activations for the output layer (see Figure 2.3). The reader can refer to Goodfellow et al. (2016) or Murphy (2022) for more details on the topic.

In practice, the network parameters are represented as matrices $N \times M \times D$, where every row N corresponds to an entry value from the previous layer $a_{i;l-1}$ (pre-activation), and every column M to a neuron unit in the current layer l $z_{i;l}$, for every dimension of the input D. Then the forward pass becomes a succession of simple mathematical operations called *floating point operations* (FLOPs). This succession

of FLOPs operations can be stored within a computational graph.

2.1.2.2 Backward pass and parameters update

At the beginning of the training the parameters are initialised randomly following some sort of Gaussian pattern distribution (initialisation of deep neural networks will be discussed in section 2.2.2.1). For the prediction to be more accurate, parameters are tuned throughout the training following a *gradient-based* optimisation scheme. Similar to classical machine learning algorithms, gradients are derived from a *loss function* $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{x_i;y_i} \boldsymbol{l}(\phi(x_i;\boldsymbol{\theta}_i)y_i)$ – also called the *cost function*, with respect to the parameters $\boldsymbol{\theta} = [\boldsymbol{w};\boldsymbol{b}]$ at each layers. The idea is to measure how far the current prediction $\hat{y} = \phi(x;\boldsymbol{\theta})$ is from the true prediction y, and update the parameter values (weights and biases) to reduce this gap. For classification tasks, the common loss function used in practice is the *cross-entropy loss*.

The backward pass consists of computing the gradients with respect to the loss for each parameter through a *backpropagation* algorithm:

$$g(\boldsymbol{\theta}) = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{2.5}$$

The *backpropagation* algorithm relies upon a directed acyclic graph to compute the individual parameter's gradient through a chain of rule of calculus: $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}\frac{\partial \boldsymbol{\theta}_i}{\partial \boldsymbol{\theta}_{i-1}}\dots\frac{\partial \boldsymbol{\theta}_{i-n}}{\partial \boldsymbol{\theta}}$. In computing this step is called **auto-differentiation** or **auto-diff** and is referred to as such in computing libraries PyTorch (Paszke et al., 2019), and TensorFlow (Abadi et al., 2015). More details can be found in deep learning books such as Goodfellow et al. (2016); Murphy (2022).

Once the gradients have been computed, the parameters can be updated following the stochastic gradient descent step:

$$\boldsymbol{\theta}_i^t = \boldsymbol{\theta}_i^{t-1} - \alpha\frac{\partial \mathcal{L}(\boldsymbol{\theta}^{t-1})}{\partial \boldsymbol{\theta}_i} \tag{2.6}$$

where $\boldsymbol{\theta}_i^t$ represents the i^{ith} parameter at time t of training, and α the learning

rate. The gradient only indicates the direction for the parameter's update, it is the learning rate that defines the step size of the update and it is probably the most important hyper-parameter in training deep neural networks (Bjorck et al., 2018; Jastrzebski et al., 2019, 2020).

2.1.2.3 Compute cost

When training a deep neural network, this cycle of *forward pass*, *backward pass* and *update* is repeated over the training data, until the network has reached convergence (the loss and gradient are null), or until the training has been stopped (early stopping or fixed number of epochs). In practice, to lessen the cost of computation, a stochastic approach (SGD) is preferred where the gradient update is computed and averaged over a small batch of training data (referred to as the **mini-batch**) instead of the entire dataset.

Training can be computationally demanding. Beyond the computational cost related to the structure of the neural network in itself – detailed in section 2.2.1.1, different components needs to be stored in memory: the mini-batch of data to be processed, the computational graph, the parameters value and their associated gradients. To speed up the training of deep neural networks, data within a mini-batch are processed in parallel. Deep neural networks rely on the processing power of specialised GPU hardware to handle the compute complexity. Chapter 3, section 3.2 will provide more insight on the matter.

2.1.3 Running inference

2.1.3.1 Compute cost

Once the deep neural network has been trained, it can be used to run inferences. In *inference* mode, only the forward pass is active as the gradients do not need to be computed anymore. Also, images do not need to be processed in mini-batch, thus the memory footprint compared to training is reduced. However, the compute complexity is still bound to the neural network architecture – i.e., the number of

parameters that are required to be stored, and the type of data being processed. Large images will generate more FLOPs operation than smaller ones.

2.1.3.2 Evaluating performance

The difficulty in training deep neural networks, as for any other machine learning algorithms, is to ensure the model generalises well to unseen data. Due to their large number of parameters, deep learning models have a tendency to over-fit (i.e., to memorise) the training data easily. But they are also tied to the inherent biases and limitations existing within the training dataset (class imbalance, missing data, etc.). Deep learning models do not extrapolate concepts but simply learn feature representations of an existing set of data.

In practice model performance is evaluated over a validation and testing data set, and computed as the mean prediction score:

$$accuracy = \frac{\sum_i (y_i = \hat{y}_i)}{\sum_i y_i} \tag{2.7}$$

where y_i is the model prediction, and $\hat{y}_i$ the true prediction. Due to the increasing use of deep learning models for real world applications, especially sensitive applications such as face recognition, or autonomous driving, *fairness* and *robustness* of deep learning has become a hot topic. Moving beyond only assessing the performance of a model based on its prediction accuracy score, researchers have tried to develop novel approaches to evaluate other aspects of deep neural networks. This topic will be further discussed in chapter 6.

2.2 Deep Learning for image processing

If processing images is a natural task for humans, it turns out to be very challenging for a computer. There is no mathematical rule that can describe what a cat looks like, or how to tell different breeds of dogs apart. One can define some features that

represent a cat's appearance, but they cannot be translated to pure mathematical rules. For a long time computer vision relied heavily on hand-crafter and engineered feature extraction tools to assist computer understanding of images (Lowe, 2004).

Deep learning has now become the dominant approach to computer vision. With its high flexibility in learning deep feature representation for images, deep learning has proven to be a powerful tool for image processing tasks. Pioneering work by Krizhevsky et al. (2012) demonstrated the great potential of deep learning models in achieving human-level performance on image classification. In over 20 years of research, computer vision has become a mature area of deep learning research with many applications in our everyday life. Led by the many industry interests such as smart cities, or autonomous drivers assistant systems, computer vision has concentrated most of the discovery around sparsity and deep learning compression. Computer vision models are also more accessible, they do not require latent memory management as NLP models do, therefore they constitute a good entry point to develop and explore strategies to reduce the size of a deep neural network.

2.2.1 Convolutional Neural Networks

MLP architectures do not have a indicative bias suited for images, as they are not invariant to translation shift.. A slight shift in the input image (translation, lighting, camera angle etc.) might result in a very different output. To overcome this issue, Convolutional Neural Networks (CNNS) were introduced with a novel spatial filtering approach called *convolutional layers*.

2.2.1.1 Building blocks

Most concepts inherent to CNNs were first introduced in the neocognitron paper by Fukushima (1980), but it is two decades later that LeCun et al. (1998) proposed the foundation of CNN with the LENET architecture. A CNN is composed of two types of building blocks: *fully-connected* layers, and *convolutional* layers. In addition extra processing or *pooling*, also known as *sub-sampling*, is usually added in-between

some layers.

Fully-connected layers *Fully-connected layers* are the core component in MLP models described in Section 2.1.1. All the inputs, x_i, are connected to every activation unit, $s_i = \phi(W_i^T x_i)$, in the layer as illustrated in Figure 2.4. If the pattern W_i is present within x_i (s_i is large positive), then the activation (ReLU) is also large. Because the catalyst of the activation directly derives from the inner product of $W_i^T x_i$, it is easy to imagine that a small change in the input can drastically impact the output (e.g., the same image with two different size resolutions, a translation of the subject in the images, etc.).

In practice, fully-connected layers, if used within CNNs, are used at the end of the neural network to aggregate the input and produce the desired output. For instance, to transform a $N \times M$ input into a 10×1 where 10 is the number of possible objects in the context of an image classification task.

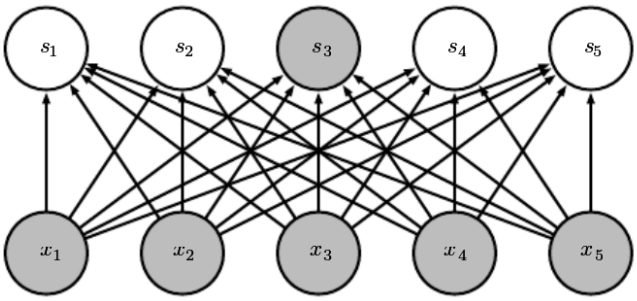


Figure 2.4: Fully connected layer where x_i are the input and s_i activation. From Goodfellow et al. (2016) *Deep Learning* book.

Convolutional layers *Convolutional layers* are an alternative to fully-connected layers that produce transitionally invariant feature detectors. The input image is decomposed and processed in small overlapping patches. Every patch of the image is compared to a small set of *filters*, also called the *kernel* (usually of size 3×3 or 5×5), to detect the presence of spatial features (or parts of objects) in the image patch. Filters are moved across the image in a sliding-window manner. Figure 2.5 illustrates the convolution operation.

For an input image, $H \times W \times C$, a convolution operation is a dot product between the kernel, $k \times k$, and patches of the input image over C input channels ($C = 1$ for greyscale images, and $C = 3$ for RGB images). To ensure that every data point is considered equally (positioned at the centre of the kernel) when performing convolution *zero-padding* can be employed to add zeros on the outside of the matrix and thus compute the convolution operation for edge values. In a similar way, a *stride* value corresponding the step size of the sliding-window can be tuned.

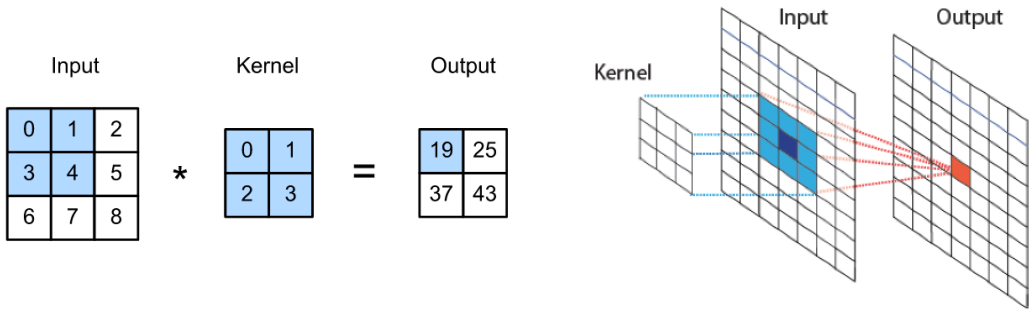


Figure 2.5: Illustration of convolution operations: (left) numerical operation, and (right) spatial filtering. Source: Murphy (2022) and (Robles, 2018)

Convolutions do not necessarily reduce the input dimension, $H \times W$, but they do increase the number of dimensions (depth) as each kernel generates a different matrix output, D, leading to $H \times W \times C \times D$. For that reason, *pooling* or *sub-sampling* is often use to concatenate the information within a filter.

FLOPs Parameters within a deep neural networks are represented by *tensors*. Reducing the size of a deep neural network is done through manipulating the dimension and elements of the different tensors of the neural network. Table 2.1 summarises the size and compute cost for different building block candidates for pruning available in a CNN.

Parameters	**FLOPs**	**Input Tensor**	**Output Tensor**	θ_i^l
Fully connected layers (Linear)				
weights	$I * O$	$I : (H_{in})$ $H_{in} = \text{input\_features}$	$O : (H_{out})$ $H_{out} = \text{output\_features}$	$[:, i]$
bias	O	–	–	–
Convolutional layers (Conv2D)				
weights	$I * O * K_w * K_h$	$I : (C_{in}, H_{in}, W_{in})$ –	$O : (C_{out}, H_{out}, W_{out})$ $H_o ut = \lfloor \frac{H_{in} - K_h + 2p}{s_h} + 1 \rfloor$ $W_{out} = \lfloor \frac{W_{in} - K_w + 2p}{s_w} + 1 \rfloor$	$[:, i, :, :]$
bias	O	–	–	–

Table 2.1: Summary of the complexity of fully-connected and convolutional layers, where I is the input, O the output, K_w and K_h are respectively the kernel width and height, C the number of channels, H and W the height and width of the tensor (input or output), p and s correspond to the *padding* and *striding* of the convolution operation.

2.2.1.2 Spotlight architectures: VGG & Resnet

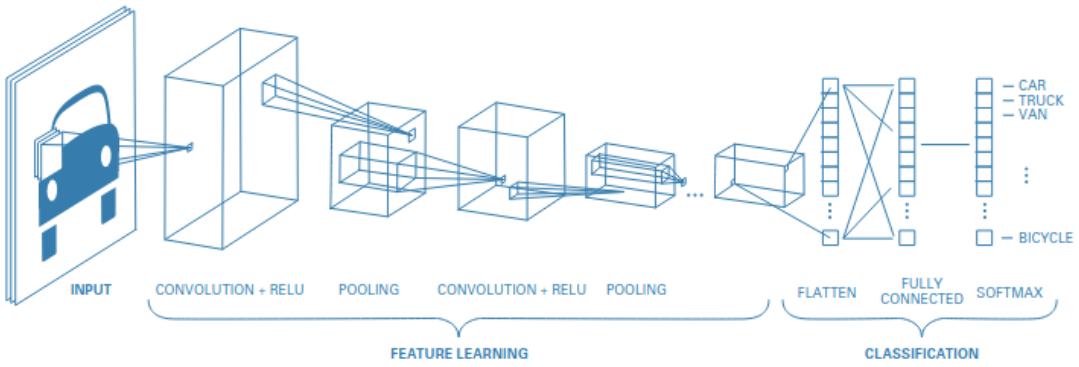


Figure 2.6: Reproduced from *Probabilitic Machine Learning* book (Murphy, 2022).

A convolutional neural network (CNN) is a succession of convolution layers with a non-linear activation, followed by pooling of the feature map. A final fully-connected layers can be added for classification output (see Figure 2.6). One of the earliest CNN model architectures (**LeNet**) was proposed by LeCun et al. (1998). It was composed of five layers: 2 convolution-pooling layers, and 3 fully connected, dense layers. The model was trained on a simple image classification task called MNIST (see section 2.3.2). However, its representation capabilities were fairly limited due to its small size. At that time training of deep neural networks was a particularly expensive process that required a lot of hand-engineering.

23 years later Krizhevsky et al. (2012) proposed a novel architecture called **AlexNet**, capable of achieving groundbreaking results on the ImageNet classification task (see section 2.3.2) and driving a renewed interest towards using deep learning models for computer vision. Their model was slightly deeper (8 layers) than LeNet with 5 convolution layers instead of 2. In addition AlexNet used ReLU activation compared to the traditional tanh or sigmoid used previously. Perhaps the major revolution of AlexNet was the use of GPU hardware to fit and train neural networks (AlexNet contains 60M parameters).

VGG architecture The compute capacity offered by GPU hardware pushed back the boundaries for deep neural architectures. Simonyan & Zisserman (2015) pro-

Figure 2.7: VGG16 architecture, where 16 represent the number of layers (convolutional and fully connected). `{filter-size} conv, {number-filter}` refers to convolutional layers, and `fc {number-neurons}` corresponds to fully connected layers. After each group of conv layers a max-pooling operation is applied, with `sizeXXX` indicating the input size for the layer group. Reproduced from

posed the first very deep CNN architecture with a range of models with varying depth from 11 up to 19 layers. They were successful in training them on large image classification tasks and were considered to be the state of the art. VGG architecture can be decomposed into four feature size groups (64, 128, 256 and 512) of small filters (3×3) stacked upon one another separated by pooling layers (see Figure 2.7). The number of features corresponds to the number of learning channels, or kernel, within a layer.

VGG architectures are amongst the most over-parameterised CNN architecture. Because layers are stacked upon each other, training VGG beyond 19 layers is extremely unlikely to converge because of gradient vanishing or exploding. A vanishing or exploding gradient phenomenon is observed when the gradient isn't flowing properly causing it to become zero or very large. This phenomenon is detailed in section 2.2.2.1). He et al. (2015) also noticed that training deeper networks resulted in higher training error and that training a plain architecture – stacked convolutions without pooling layers in between – with 34 layers has a higher error than a plain 18-layer model, even though the 18-layer network is a subset of its deeper counter-part.

ResNet architecture To train very deep networks beyond 19 layers, He et al. (2015) proposed adding skip connections to prevent degradation of the signal. A

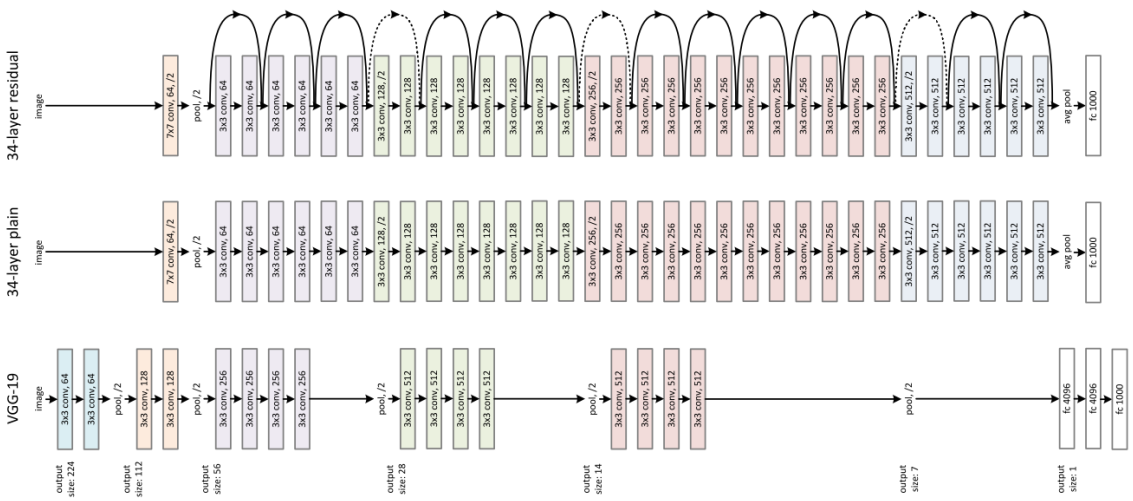


Figure 2.8: Illustration of different CNN architectures: ResNet-34 (top), plain-34 (middle) and VGG-19 (bottom). Skip connections allows the deep neural network to be trained beyond 34 layers, dashed residual connections represent `CONV-1x1` from succession of two layers that differ in size. Source He et al. (2015).

skip connection – also called a *residual connection* – is a simple mapping of the input signal from the previous layer into the output signal of the current layer (see Figure 2.9). It enables the gradient to be better propagated throughout the network (Balduzzi et al., 2018) and helps smooth the optimisation landscape (Li et al., 2018). Note that the residual connection is a simple identity mapping of the input, or a 1×1 convolution when the size between the input and output signal differs. Figure 2.8 presents different CNN architectures side-by-side.

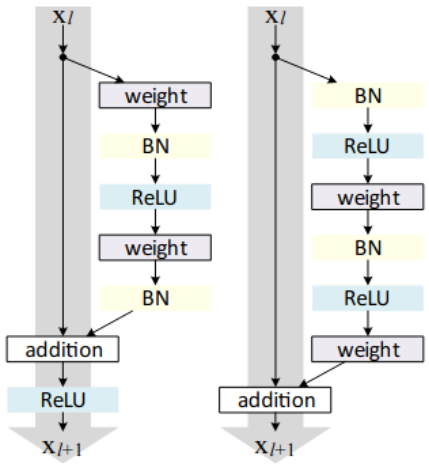


Figure 2.9: Illustration of a residual blocks with a skip connection: (left) *post-activation* ResNet, and (right) *pre-activation* as in PreAct ResNet Source: He et al. (2016b)

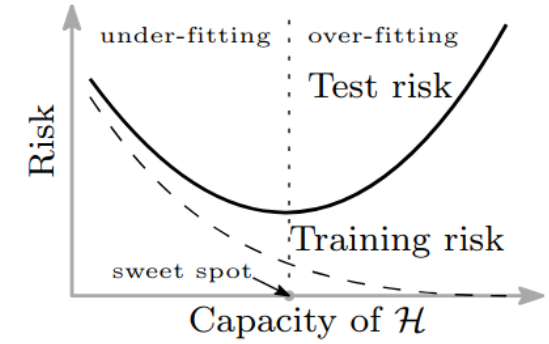
There are two ways to connect layers with identity mapping, either after the non-linearity *post-activation* as originally designed in He et al. (2015) – after the block CONV-RELU-BN block –, or before the the activation unit *pre-activation* as proposed in He et al. (2016b) – after the block RELU-BN-CONV (see Figure 2.9). The later is called a **PreActivation ResNet** (He et al., 2016b) and obtains better performance than ResNet models.

2.2.2 Over-parameterisation

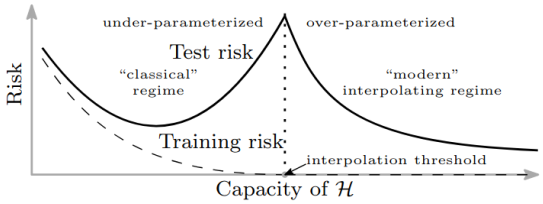
One of the fundamental when designing machine learning models is to avoid *big* models all cost. The challenges of having more parameters than training data points is to over-fit the training set, meaning that the model will memorise the training examples instead of learning to solve the task, resulting in poor model generalisation. This phenomenon is referred to as the *variance-bias* trade-off (see Figure 2.10a). A model should incorporate the right amount of parameters, to avoid producing highly biased predictions (under-fitting, not enough parameters), and being too sensitive to the variance within training examples (over-fitting, too much parameters). There should exists an optimal capacity where that the model minimise both type of errors leading to better prediction over unseen data point (generalisation).

Pruning is a compression approach that reduces the complexity of a machine learning model by removing the parts that are non-critical to produce the outcome decision of the algorithm, and thus reduces over-parametrisation. This concept was first introduced for decision-trees algorithms which are prone to over-fit the training data due to their high parameterisation, and thus, by pruning part of the tree, memorisation can be reduced leading to a better balance bias variance trade-off (Quinlan, 1987).

But if this statement was true for classical machine learning, deep learning has proven that for modern algorithms, larger models are better. The strength of deep learning models arise from the particularity of them to be inherently hugely over-parametrised. Datasets often contain between 60k images for smaller datasets such



(a) Classical regime



(b) Modern regime

Figure 2.10: Bias-variance trade-off. Reproduce from Belkin et al. (2019)

as CIFAR10 (Krizhevsky et al., 2009), up to 1.3M images for larger ones such as ImageNet (Deng et al., 2009). Modern CNN architectures like ResNet50 or VGG11 have roughly 25.6M and 129M parameters respectively. Note that recent approaches to computer vision use transformer architecture type, state-of-the-art models CvT (Wu et al., 2021), ViT (Yuan et al., 2021) and BiT (Kolesnikov et al., 2020) that contain at most 227M, 307M and 920M parameters respectively.

This phenomenon comes from a *double descent* that arise from being in an over-parameterised regime (Belkin et al., 2019; Nakkiran et al., 2021). Because deep learning models are trained to fit the data (zero training error), called the *interpolating regime*, the more parameters, the better the performances (see Figure 2.10b).

To understand if and how the number of parameters can be reduced, it is necessary to understand why over-parameterised model do learn better than they smaller compact counter-part. They are two intuitions to explain why over-parameterisation helps deep neural networks to learn complex tasks: (1) from an optimisation perspective, over-parameterisation helps SGD to find a good minima by smoothing the optimisation landscape, and (2) the more trainable parameters, the more features

we can learn and extract from the image and thus over-parameterisation allows greater knowledge representation. However, over-parameterisation comes at the cost of heavy regularisation techniques to prevent negative impact on generalisation performance (Zhang et al., 2021). Large models can be tricky to optimise with risks of vanishing or exploding gradients, on top of requiring extra regularisation to generalise properly.

2.2.2.1 Vanishing Exploding gradients

When training large, and more especially very deep neural networks, the gradient may vanish or explode (Hochreiter et al., 2003). Every time the signal passes through an activation unit the gradient is either diminished or amplified and the deeper the network the more at risk the gradient is of catastrophic increase or decrease. It goes without saying that the deep neural networks cannot learn when gradient vanishing or exploding occurs.

Activation functions can prevent this from happening by bounding the gradient into a $[0, 1]$ window. A good initialisation strategy is also important as large, and respectively small, initial parameters value will push the gradient towards one extreme or the other. In practice parameters in a neural network are initialised following a Gaussian distribution and designed such that the mean activation is zero and the variance across layers is the same. This ensures the gradient flows properly through the network and is called the *Xavier* initialisation scheme (Glorot & Bengio, 2010):

$$W^{[l]} \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}} \tag{2.8}$$

$$b^{[l]} = 0 \tag{2.9}$$

where $n^[l-1]$ corresponds to the number of parameters in the previous layer. Xavier initialisation was developed around sigmoid and tanh activation units, He et al. (2015) proposed a novel initialisation scheme called *Kaiming* initialisation that bet-

ter suits ReLU/leaky-ReLU non-linearity activation.

The initialisation of deep neural networks has always been of importance as it defines the starting point of the SGD optimisation. Ill-conditioning of the network directly impairs the performance of the model. It is an important point to keep in mind as parameters can be removed from the network anytime during training. Thus, when pruning the network it is essential to make sure that the network structure is well-preserved to ensure good re-trainability.

Another way to prevent the gradient from vanishing or exploding is the use of residual connections to enable the gradient to flow directly between layers as discussed previously in section 2.2.1.2.

2.2.2.2 Leveraging over-parameterisation with regularisation

If a good choice of activation function and initialisation scheme can ensure a good flow of the gradient for the early stage of training, it does not prevent the network from over-fitting the training data in the long term. The key in training deep neural networks relies in their heavy use of regularisation techniques to help the network learn despite being over-parameterised.

Norm penalty and weight decay A straightforward regularisation technique in machine learning to limit the model capacity is to add a norm penalty to the objective or loss function to constrain the parameters from deviating too much. In deep learning, it is common to use a l_2 norm regularisation, known as *weight decay*. This can be written as:

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{2.10}$$

where α is an hyper-parameter defining the strength of the regularisation – in practice, α is of order of magnitude $1e^{-5}$. Note that the change in biases only induce little variance over the model performance, therefore only the weights $\boldsymbol{w}$ are consid-

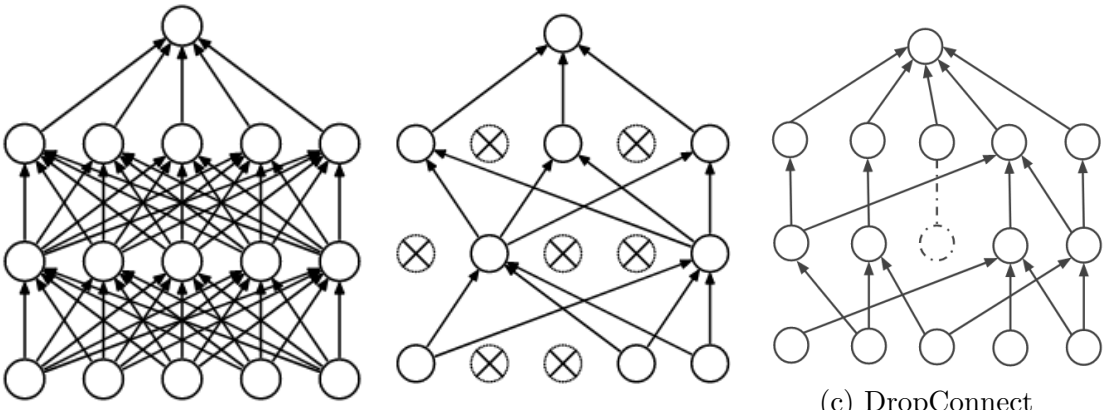
ered for regularisation (Goodfellow et al., 2016). l_2 norm regularisation, or weight decay, can be interpreted as preserving the parameters alongside the dimension that contributes to reducing the objective or loss function the most and discarding the ones that contribute poorly.

Early Stopping Another simple way to prevent the training data from over-fitting is to apply an *early-stopping* strategy. The training is stopped when the loss is no longer decaying after a long period of time, or when the training error starts to rise up again. This kind of approach enables networks to learn from generic features that generalise to unseen data (Achille et al., 2020).

Dropout and sparsity Fully-connected layers have a tendency to easily over-fit the training data due to neurons co-adapting. Srivastava et al. (2014) proposed a *Dropout* regularisation method that randomly drops compute units during the training to prevent over-fitting. Every forward pass a different set of neurons is randomly masked from the network to force every unit to learn independently from their neighbours. An extension over Dropout, DropConnect (Wan et al., 2013), proposed a similar mechanism masking the connections instead of the neuron units (see Figure 2.11).

More generally, inducing sparsity in a neural network can work as a strong regulariser. When all the parameters in the models are active (Figure 2.11a), the model is said to be *dense*. When parts of it are removed, either by groups (Figure 2.11a), or individually (Figure 2.11a), the model is said to be *sparse*. In both, Dropout and DropConnect, the sparsity is only temporary to prevent the network from over-fitting. In pruning, the goal is to make this sparsity permanent to reduce the size of the deep neural network. This concept of sparsity is the main subject of study of this thesis and will be detailed throughout the coming chapters.

Batch normalisation Because the data are processed in mini-batches, deep networks suffer from *covariate shift* induced by the data distribution changes between



(a) Dense network (b) Dropout (c) DropConnect

Figure 2.11: Illustration of different type of *drop-like* regularisation for a MLP network. While removing neurons (b) automatically removes all associated weights, removing weights does not necessarily shut down the associated neuron. However, if all the incoming connections (or weights) are removed as pictured in (c) middle-line, removing all the connections is equivalent to removing a neuron.

different mini-batches (Shimodaira, 2000). A popular and powerful approach to overcome this issue is *Batch Normalisation* (Ioffe & Szegedy, 2015). Batch normalisation is an extra layer that is added to standardise and shift the input signal, ensuring that across all mini-batches the previous layer has zero mean activation and unit variance:

$$z_{out} = \boldsymbol{\gamma} \odot \hat{z}_n + \boldsymbol{\beta} \tag{2.11}$$

$$\hat{z}_n = \frac{z_{in} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.12}$$

where B is the mini-batch, μ_B and σ_B^2 are respectively the mean and variance of the minibatch, z_{in} the input signal, $\hat{z}_n$ the standardised input signal for the mini-batch, and z_{out} the normalised input signal across all mini-batch scale by a factor γ and shifted by a factor β, γ and β being learnable parameters. For convolutional layers, batch normalisation observes a channel-wise (ie. filters-wise) scaling/shifting of the parameters. When the model is deployed in production, it often processes a single image at a time, thus mini-batch statistics cannot be computed. During training, the moving average of all mini-batches, μ_l and σ_l^2, are computed. Those statistics

are then used in equation 2.12 to replace μ_B and σ_B^2 when the model is used for inference.

In CNNs, the spatial filtering and depth of the network diminish greatly the regularisation effect of Dropout, and DropConnect techniques. The benefits of using batch normalisation layers in CNN, while not fully understood yet, drastically improve the model performance reducing its sensibility to learning-rate and accelerating training-time (Santurkar et al., 2018; Arora et al., 2018). There exists other forms of normalisation – *layer normalisation* (Ba et al., 2016), *group normalisation* (Wu & He, 2018), *instance normalisation* (Ulyanov et al., 2017)— but they are outside the scope of this study. In practice batch normalisation is the most common form of regularisation used for computer vision models.

2.3 Scope of this thesis

Deep neural networks are highly over-parameterised resulting in high computational cost in terms of resources and memory storage, impacting the environmental footprint and universal accessibility of deep learning technology (Strubell et al., 2020; Thompson et al., 2020). Inducing sparsity, ie. removing parameters from the neural networks, can help reduce the compute requirement required to run deep networks.

This thesis explores unstructured pruning, a compression method that aims to remove individual parameters (weights) based on their importance upon the dense network function. The core study focuses on importance measures used to estimate the usefulness of a parameter, and evaluates what characteristics make a good pruning criteria.

2.3.1 Model evolution dynamics

To understand how to evaluate the importance of a parameter, it is important to understand how they evolve throughout the training. Trainable parameters are referred to as $\boldsymbol{\theta} = \{\boldsymbol{W}; \boldsymbol{b}\}$ – trainable parameters associated with batch normalisation

regularisation are omitted, with $\theta_l = \{W_l; b_l\}$ corresponding to a specific set of parameters at layer l.

Weights, $\boldsymbol{W}$, represents how two variables interact with one another and emphasises the strength of the connection between two variables. Biases, $\boldsymbol{b}$, on the other hand do not induce much variance as they only control one activation unit at a time. Thus when removing parameters from a neural network, efforts are primarily concentrated on weights – biases are ignored.

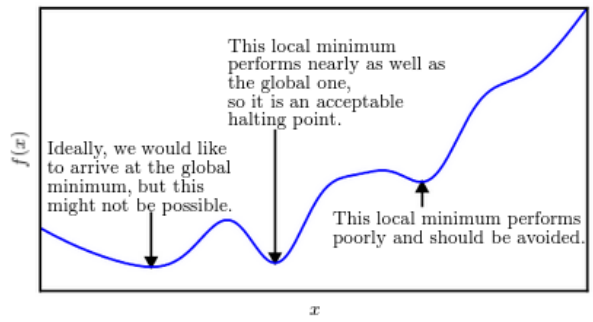


Figure 2.12: Illustration of the minimisation problem that the gradient descent algorithm is trying to solve, where $f(x)$ represents the loss function. Reproduced from the Deep Learning book Goodfellow et al. (2016).

Objective function When training a deep neural network, parameters are optimised through gradient descent in order to minimise a loss function $\mathcal{L}$. It can be interpreted as finding a good local minima within the optimisation landscape, also called the loss landscape, such that it is not possible to further reduce the loss function (see Figure 2.12). The underlying properties and geometrical shape of the loss landscape is unknown, but the loss function provides information about the curvature of the landscape at a specific point, and help finding a suitable local minima through stochastic gradient descent optimisation.

For classification tasks the common loss used in practice is the cross-entropy loss, but for any type of loss it can be approximated using a Taylor expansion. Approximating the loss around $\boldsymbol{\theta}_0$ can be written as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T H(\boldsymbol{\theta})(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \mathcal{O}((\boldsymbol{\theta} - \boldsymbol{\theta}_0)^3) \tag{2.13}$$

where $\nabla \mathcal{L}(\boldsymbol{\theta})$ is the first order derivative, also known as the Jacobian, and $H(\boldsymbol{\theta})$ is the second order derivative, also known as the Hessian. The **Jacobian**, $\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$, contains the gradient of the different parameters and indicates in which direction to move to make improvement on the loss function, thus by taking a step in the negative gradient direction, it decreases the loss.

The **Hessian**, $H(\boldsymbol{\theta}) = \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}^2}$, can be interpreted as being the Jacobian of the gradient and provides information about the curvature of the loss function. More precisely it indicates whether taking a step ϵ in the direction of the gradient is going to be smaller (positive curvature), equal (no curvature), or bigger (negative curvature), and thus measure the magnitude contribution of a parameter to decreasing the loss function.

After a step-size update $\boldsymbol{\theta}$ becomes $\boldsymbol{\theta}_0 - \epsilon \boldsymbol{g}$, where $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ and ϵ is the learning rate. Replacing $\boldsymbol{\theta}$ in equation 2.14, we obtain:

$$\mathcal{L}(\boldsymbol{\theta}_0 - \epsilon \boldsymbol{g}) = \mathcal{L}(\boldsymbol{\theta}_0) - \underbrace{\epsilon \boldsymbol{g}^T \boldsymbol{g}}_{\text{A}} + \underbrace{\frac{1}{2}\epsilon^2 \boldsymbol{g}^T H \boldsymbol{g}}_{\text{B}} + \mathcal{O}^3 \tag{2.14}$$

with **A** being the expected improvement following the slope of the function, and **B** the correction to account for the curvature of the slope. Note that for this approximation to be correct, small update steps are necessary. Also, even for small MLP, computing the Hessian is intractable. In practice it is common to use a Gauss-Newton approximation to compute the Hessian (Schraudolph, 2002):

$$\mathbf{H}(\boldsymbol{\theta}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \frac{\partial f_{\boldsymbol{\theta}}(x_i)}{\partial \boldsymbol{\theta}}^{\top} \nabla^2_{u=f_{\boldsymbol{\theta}}(x_i)} \ell(u, t_i) \frac{\partial f_{\boldsymbol{\theta}}(x_i)}{\partial \boldsymbol{\theta}}}_{\mathbf{G}(\boldsymbol{\theta}),\ \text{the Generalized Gauss-Newton}} + \underbrace{\sum_{k}^{K} \frac{\partial \ell(u, t_i)}{\partial u_k}\Big|_{u=f_{\boldsymbol{\theta}}(x_i)} \frac{\partial^2 f_{\boldsymbol{\theta}}(x_i)_k}{\partial \boldsymbol{\theta}^2}}_{\approx 0} \tag{2.15}$$

$$\approx \mathbf{G}(\boldsymbol{\theta}) \tag{2.16}$$

Momentum If the region of the loss landscape along which the gradient is travelling is flat, or if it falls in a ravine, optimising through gradient descent can be very slow as it progresses either slowly or oscillates between the ravine walls. A simple trick to help with the optimisation is to add momentum, also called Nesterov momentum after its creator (Nesterov, 1983). Momentum works as an exponential moving average of the past gradients applied as weight upon the current step (see Figure 2.13):

$$\upsilon_t = \gamma \upsilon_{t-1} + \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \upsilon_t$$

where υ is the exponential moving average of the past gradient, and γ the weight factor (in practice $\gamma = 0.9$).

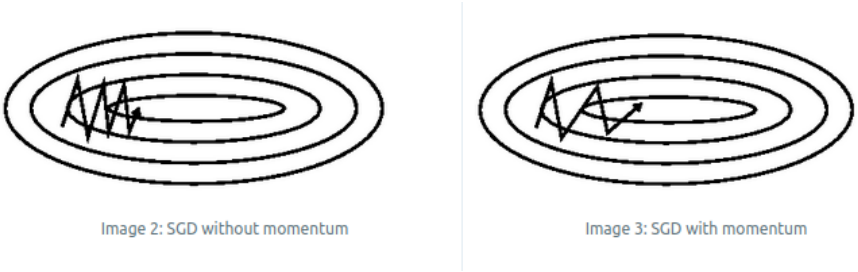


Figure 2.13: Momentum update. Taken from Sebastian Ruder's blog article *Optimizing Gradient Descent* (https://ruder.io/optimizing-gradient-descent).

(a) MNIST (b) CIFAR10

Figure 2.14: Sample images of three different datasets: (a) MNIST, (b) CIFAR10.

2.3.2 Datasets

In this thesis we consider the use case of image classification task. Image classification has long served as an academic benchmark for empirical theoretical research and thus most recent research in the pruning literature used similar datasets (Frankle & Carbin, 2018; Lee et al., 2019; He et al., 2016a; LeCun et al., 1989).We use the same three academic benchmark datasets to assess the performance of the different pruning methods investigated:

- **MNIST** (LeCun et al., 1998): MNIST dataset is a handwritten digits dataset containing grey-scale images of fixed size image 28×28. The different digits have been centred, the dataset contains 10 classes (0 to 9 digit), 60, 000 training images and 10, 000 test images (see Figure 2.14).

- **CIFAR10** (Krizhevsky et al., 2009): CIFAR10 dataset is an image classification dataset containing 60, 000 32×32 color images (50, 000 images for training, and 10, 000 for testing) (see Figure 2.14).

- **ImageNet** (Deng et al., 2009; Russakovsky et al., 2015): The ImageNet dataset is the most predominant image classification dataset for large scale and real world applications. It contains 100,000 sets of synonyms, each represented by 1,000 images.

Because of their small-scale nature, both MNIST and CIFAR10 datasets are reasonably inexpensive to train to better stress the limitations of pruning methods, which makes them ideal for conducting empirical research. On the other hand,

ImageNet is used in real-world AI applications and constitutes a good candidate to assess the scalability of our discovery once the preliminary stage of empirical exploration is done.

2.3.3 Model architectures

Similarly, different model architectures are tested to assess the robustness of the results presented across different scales. We picked three model architectures that are commonly found the in pruning literature as they offer a variety of complexity in terms of computational operations (FLOPs), with a wide range of different model capacity (overall number of parameters):

- **MLP** – small capacity, low complexity – Presented in Section 2.1, MLP is the simplest form of deep neural network as it is composed of only fully-connected layers. It has a fairly low number of parameters (over a 100k), allowing to better compute model statistics.

- **VGG** – large capacity, medium complexity – presented in Section 2.2.1.2. Hugely over-parameterised, VGG model architectures are known to be easy to compress. Despite their high number of parameters (>100M), they remain fairly simple as convolution blocks are stacked up on top of each other with no extra compute. They are a deeper version of MLP with convolutional operations.

- **ResNet - PreAct-ResNet**: large capacity, high complexity – presented in Section 2.2.1.2. Despite being less over-parameterised than VGG models (<100M parameters), ResNet, and respectively PreAct-ResNet, are more complex to compress due to their skip connections.

Considering those three network architectures offer a variety of use-cases to conduct our empirical analysis on. This will allow us to assess whether the size of the model (capacity) or the number of FLOPs in the architecture (complexity) has an impact on the pruning outcome. Note that these models would be considered to be

small-scale compared to the most recent architectures used in computer vision, but if the pruning hypothesis tested do not scale properly across our three models, it is unlikely they will succeed for larger models.

2.4 Chapter Summary

In this chapter, the foundations of deep neural networks were reviewed. Through the mainstream XOR examples in Section 2.1.1, I demonstrated how a simple Multi Layers Perceptron (MLP) could be used to combine neural units to model non-linear boundaries. This property of deep neural networks to fire non-linear activation $z_l = \boldsymbol{W} z_{l-1} + \boldsymbol{b}$ based on an input signal has been exploited to solve complex tasks such as image processing. However, a hand-crafted approach where all the parameters $\boldsymbol{\theta} = \boldsymbol{W}; \boldsymbol{b}$ would be carefully tuned is not feasible as modern deep learning architectures far exceed 1M parameters.

To learn those parameters, deep neural networks are trained following a gradient descent optimisation scheme. The different parts of training – forward pass, backward pass and update – were presented in Section 2.1.2 introducing the core elements that comprise a deep neural network architecture: the choice of activation function to induce non-linearity, the loss function to measure the prediction error, and the optimiser to find the best set of parameters. Concepts for measuring the compute cost in deep learning models with Floating Point Operations (FLOPs) were briefly introduced in Sections 2.1.2.3 and 2.1.3.1, this factor will be further discussed in Chapter 3.

At the beginning of the deep learning era, computer vision was the predominant use-case application around which research was concentrated. It nurtured many great discoveries and has today many application in our everyday life (smartphones, image retrieval, etc.). In Section 2.2.1 Convolutional Neural Networks (CNNs) were presented alongside the neural network architectures – VGG and ResNet – studied in this thesis. CNNs take advantage of the high feature space and non-linearity of deep neural networks to learn spatially invariant features to detect objects, or

elements in an image. However for them to learn and extract useful knowledge, the models are hugely over-parameterised

In Section 2.2.2, the implication of over-parameterisation on learning were presented. On top of potential threats to over-fit the training data, the activation units put the gradient signal at risk to be either diminished until vanishing or amplified and exploding, leading to the network being unable to learn anything. Many methods can be applied to stabilise the learning: careful tuning of the learning rate, early stopping strategy and adding regularisation (weight decay, dropout, batch normalisation).

Over-parameterisation is not only a challenge for optimisation; over-parameterisation also alters the computational footprint of deep neural networks hindering their deployment on constrained and non-GPU devices. The scope of this thesis explores how inducing sparsity, ie. removing unimportant weights, can create more efficient neural networks. In Section 2.3.1 more insights on the evolution of weight's values throughout training were provided, presenting more detail behind the optimisation procedure of deep learning models.

Finally in Section 2.3.2 and Section 2.3.3, a list of the different datasets and models utilised throughout this thesis were presented.

2.5 Chapter conclusion

Because of the increasing implication of computer vision applications driven by deep learning in real-world settings (smartphones, video surveillance, photography, etc.), working towards more efficient AI has been of central importance. Deep learning models are hugely over-parameterised with repercussions on the computational and environmental footprint for deploying such models, as well as universal accessibility of deep learning technologies. To reduce the amount of parameters in deep neural networks, it is possible to *sparsify* the network by removing unimportant connection, or weights. Throughout this thesis, I will present and review unstructured pruning methods with a focus on how to measure the importance of parameters and what

are the implications for removing parameters in a deep neural network.

Chapter 3

Internet-of-Things and Applied AI

Deep learning has emerged as a powerful tool to handle complex decision-making achieving human-level prediction, especially with media data such as images, videos and natural language (Bengio, 2009; Krizhevsky et al., 2012). The progress in AI and the increase in data collection have greatly accelerated the development of applications in real-world settings such as face recognition or autonomous driver systems.

Over the last two decades, Internet-of-Things (IoT) has extended computing capacity beyond traditional desktops and servers, connecting the real and virtual together through smart devices (Gubbi et al., 2013). Nowadays most people possess one or more smart devices: smartphones, smart-TVs, etc. This provides a distributed environment of interconnected devices able to communicate and exchange data with one another to enable environment awareness and data-driven decision making. However, the broad adoption of AI applications in the IoT setting has not yet been realised. The amount of data generated by IoT devices – in 2014 this number was estimated as 233 exabytes and by 2020 this number was set to exceed 1.600 exabytes (Markkanen, 2015) – can greatly benefit deep learning, but it raises concern about the environmental and societal impact of smart technologies. Deep learning models require heavy compute power to run inference, while IoT often captures highly sensitive data that should be processed as soon as possible to reduce privacy threats. Therefore, particularly with the additional load of video data, ef-

ficient management of data is critical for successful deployment in smart cities and smart ecosystems.

In this chapter, we will present a use-case application of video analytics in a smart city scenario. Different challenges encountered, from computational power to efficient data management, will be presented and ways to overcome them will be proposed. We will also briefly review the role hardware capacity plays in the real-world adoption of AI applications.

3.1 Use-case: Crowd monitoring for Smart City

Section 3.1 has been published in Ballas et al. (2018)

Smarter cities utilising Internet-of-Things (IoT) technologies are required to provide a sustainable environment to accommodate the needs of the increasingly urban population of tomorrow and preserve natural resources (United Nations. Department of Economic and Social Affairs. Population Division., 2016). Various applications that rely on image and video processing can highly benefit planning and logistics around the city. However, because cities are big and public areas, deploying and testing such technologies at scale represents a big challenge to manage the quantity and variety of potential sensor data including video. Three major issues encountered are (1) network bandwidth; (2) real-time responsiveness and (3) preserving personal data privacy.

Crowd monitoring using video analytics can provide excellent real-time information on crowd density or abnormal situations, but it requires a fast turnaround of processing. Moreover, monitoring public spaces also implies potential intrusiveness of surveillance video systems and, as public awareness is growing, legislation is being enacted to protect personal data requiring systems to avoid the capture, transmission and storage of data where individuals are visible. Therefore, it is crucial to emphasise a security-by-design approach, where only the minimum useful data are captured, processed and potentially stored. Therefore, processing video images at the point of capture is crucial for more efficient data management as only numerical

data (the outcome of the prediction) needs to be transferred from the edge to the centre of control. In the following section, we will see how the challenges of deep learning applications impact these requirements.

3.1.1 Croke Park: Smart Stadium for Smarter living

Cities are public areas and testing or deploying solutions can be highly challenging. In a stadium, we encounter similar scenarios as in a city. On match days, a large crowd is gathered in a stadium requiring precise logistics to manage the huge flow of people, prevent congestion and provide a quick response in case any incident occurs. Croke Park stadium in Ireland for instance can host up to 80k people at maximum capacity, that number can be met during major sports events or open-air concerts. However, a stadium remains a controlled environment where sensors and data collection can be easily managed and monitored. For that reason, a stadium is a perfect place, small enough to try but large enough to stress out smart cities applications.

Those were the motives that led to the creation of the Smart Stadium for Smarter Living project initiative. This project was initiated as a collaboration between Microsoft Ireland, Intel Ireland, Dublin City University and Croke Park Stadium, to provide within Croke Park stadium a test-bed to try-out smart cities technologies at scale. Many videos and sound capture sensors were put at disposition, in addition to low-compute edge devices to mimic a real city environment and test-out deployment of smart applications (see Figure 3.1).

One of the major security concerns in stadiums is unexpected crowd behaviour movement. During a matchday, at break time a large flow of people converge to restoration areas or bathrooms causing congestion, long queuing and increasing the threat of hazardous crowd behaviour. Monitoring crowd density can help understanding specific patterns to ensure people move in a safe, secure and predictable manner, this has the potential to improve the quality of experience but also, critically, crowd safety.

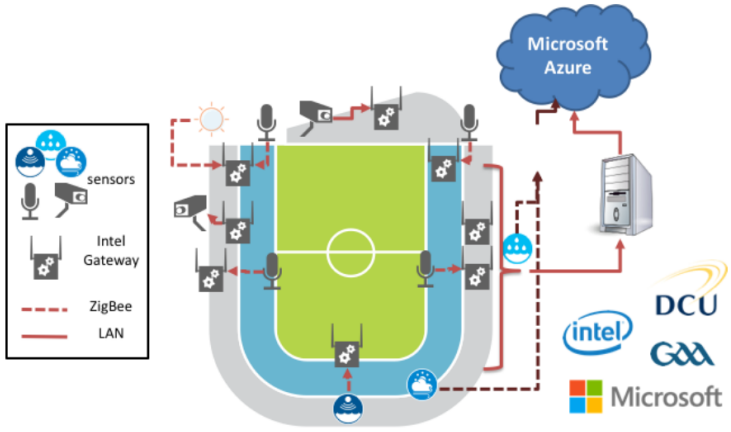


Figure 3.1: Croke Park smart stadium sensors map.

Crowd monitoring uses sensitive images that potentially contain personal data (faces). Outsourcing storage and processing to a third party is not suitable. When performing crowd analytics only metadata such as the crowd density, activity or the number of people present is generally of interest. The image in itself is not necessary. Therefore, video processing at the edge improves responsiveness while also ensuring better data privacy. But how feasible is it to process deep learning on constrained devices?

3.1.1.1 Fog computing

When it comes to IoT applications, it is crucial to emphasise a security-by-design approach, meaning only useful bits of information should be captured and stored. With the large volume of data generated by smart devices a natural platform to handle analytics for IoT applications is *cloud computing*. It presents many advantages as it has a highly scalable compute capacity. However, using cloud services implies offloading some workload and data to a third party client, which can cause issues in terms of data privacy and reliability. This is particularly true with sensitive data, such as video data where people can be easily identified, with new European regulation 2016/679 (REG) preventing improperly anonymised data from being stored. In that case, performing the analytics at the point of capture, at the *edge*, and only

extracting meaningful information is recommended.

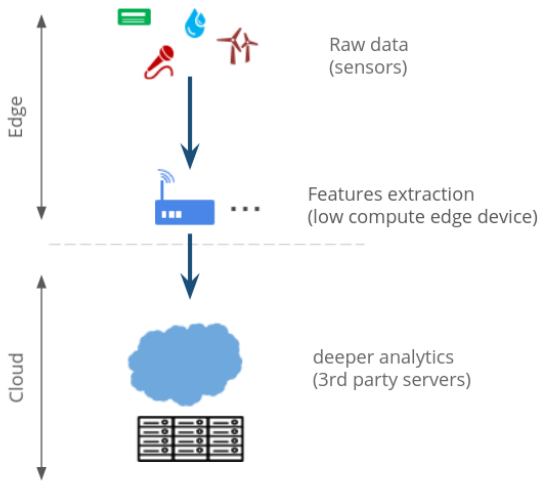


Figure 3.2: Fog computing paradigm

This paradigm is called *fog* or *edge computing*. One of the main drawbacks of cloud computing is dealing with real-time applications. In Verma et al. (2017) the authors have surveyed network topologies for real-time application and state that in their current form, data centres are not suitable for real-time processing due to network lag and transfer delays. This was also identified in the Smart Stadium project during an exercise to measure crowd sound on busy match days (Little et al., 2017).

In Bonomi et al. (2014), the authors presented an alternative to cloud-centric applications that they named *fog computing*. The pipeline they proposed uses the edge of the network to alleviate cloud limitation by carrying out some of the processing on the devices themselves (see Figure 3.2). This enhances more robust real-time applications and leverages the dependency on cloud computing. Gateways can be used to carry part of the workload and communicate with other instances. Gateways are minimal computers, similar to Raspberry Pi, where the emphasis is put on the connectivity and robustness to poor weather rather than compute power.

3.1.2 Performance of video-processing at the edge

To assess the proficiency of real-time video processing at the edge and understand the limitations of processing data close to the point of capture, we test three computer vision algorithms of different complexity. All three algorithms are inspired from a crowd monitoring use-case scenario and implemented using Python using computer vision and deep learning libraries (see Table 3.1 and Appendix A.1 for more details on hardware specifications). The different algorithms tested are:

Crowd density: A simple algorithm to estimate the density of the crowd from video footage. The frames are ingested and converted to greyscale images to extract the background of the scene and compute interesting point locations corresponding to objects in the scene. The crowd density is estimated by splitting the frame into a grid and computing the percentage of grid cells that are occupied by one or more tracked points. This algorithm has low complexity.

ResNet50: Object detection is an integral part of image processing and is known to be computationally demanding. Residual network architecture has been introduced by He et al. He et al. (2016a) for object recognition. It is among the best performing state-of-the-art deep learning models for image classification. We use it as a baseline for deep-learning-based algorithms. It represents a medium complexity, due to its low-resolution input images.

Crowd Counting: Finally, we test an advanced crowd monitoring algorithm design to count the number of people. ResnetCrowd was developed by Marsden et al. (2016) to achieve crowd counting. It is based on a ResNet50 He et al. (2016a) architecture and uses a heat map approach to estimate the number of people in a scene. High-resolution images are divided into patches and fed to the ResnetCrowd to count the number of people. Each individual result is summed to compute total crowd counting. This model achieves state-of-the-art results and has high complexity with significant compute requirements.

We assess the responsiveness of the different algorithms on two CPU machines,

Name	Description	Complexity	Deep Learning	Data Size
`(i)` **Crowd Density Estimation**	Extract background of image to estimate the density of people in the scene	Low	No	1024×728
`(ii)` **ResNet50 from Keras**	ResNet50 default Keras model application with ImageNet weights	Medium	Yes	224x224
`(iii)` **Crowd Counting**	Crowd counting from Keras ResNet50 model with custom weights (Marsden et al., 2017)	High	Yes	1024×728

Table 3.1: Crowd monitoring algorithms tested to assess compute power abilities at the edge. The complexity is measured as the number of compute operations required to process one data point and is arbitrarily expressed to compare the different algorithms with one another.

(1) an Intel *Atom E3825*, typical for edge gateway devices, and (2) an Intel I5-3210M CPU. The main differences between the two CPUs are highlighted in Appendix A.1. The software was implemented in Python using a multi-threaded approach to ingest raw images from an emulated IP camera. Images are pre-processed frame by frame and the meta-data – density or crowd number for algorithms `(i)` and `(iii)`, and the object class for algorithm `(ii)` – is sent to the cloud. Note that we do not transfer or store video images. The run-time was monitored with the `timeit` library.

The IP camera was emulated with a Python RESTful web service sending images every second using the HTTP protocol. For the Crowd Density estimation and Crowd counting algorithms, the data consisted of 10 images with high resolution (1024×728 pixels) extracted from the Shanghaitech Part_B dataset Zhang et al. (2016). The Shanghai street scenes contain between 23 people and 476 people. For the ResNet50 model, 10 images from the ImageNet dataset Deng et al. (2009); Russakovsky et al. (2015) with low resolution (224×224 pixels) were used. The results are presented in Table 3.2

Algorithm	I5-3210M CPU		Atom E3825 CPU	
	Fps	Spf	Fps	Spf
Crowd Density	12.5	0.08	6.7	0.15
ResNet50	2	0.5	0.25	4
Crowd Counting	0.02	44	0.005	219

Table 3.2: Performance comparison reported as Frames-per-second (Fps) and Seconds to process frame (Spf) for both hardware devices tested. The **I5-3210M CPU** is between twice to four time more efficient at running crown monitoring algorithms than the **Atom E3825 CPU**.

3.1.2.1 Results

Basic image pre-processing workload can easily be handled by low compute devices like gateway devices. This is the case for the crowd density algorithm that doesn't require advanced computations. However when the complexity of the algorithm increases there is a significant difference between the two CPU machines. It shifts from an order of magnitude of $\times 2$ to $\times 4$.

This increased difference in performance can be explained by (1) the increased complexity, and (2) the memory requirement. More complex algorithms usually imply more mathematical operations also called FLOPs (floating-point operations). This has a direct impact on the processing time as it is bound to the device compute power, the CPU in our case. Because CPUs are in charge of sequencing logic tasks, they have only a limited amount of power dedicated to pure mathematical computation. That explains the big difference between basic image processing and deep learning algorithms. Note that the high number of FLOPs induced with deep neural networks is the reason why GPUs are used for training and inference. Therefore having only CPU power available at inference time is still a major bottleneck for executing advanced deep learning applications. The other cause for slowing the frame-per-second rate is the size of the image we want to process for two reasons. First, a larger image implies more FLOPs as every pixel needs to be processed. Second, it will flood the memory which can greatly reduce the speed of processing images and the algorithm cannot fit into the memory at once. In Figure 3.3 we show how reducing the input size accelerates the processing time of the crowd counting

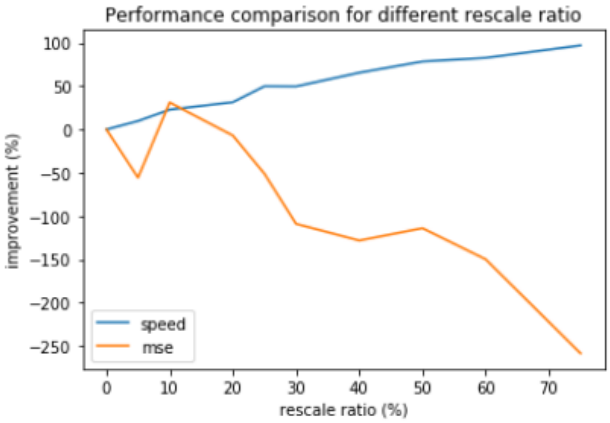


Figure 3.3: Impact on reducing the image input size on the speed. The bigger the data input size, the higher the number of compute operation will be and thus the algorithm will require more compute power.

algorithm. The bigger the data input size, the higher the number of pixels in the input image will be and thus the number of mathematical operations required to process the image will increase. However, some algorithms are bound to process only a specific size of data input, moreover by decreasing the size of the input data, we also decrease the quality of the image resulting in a loss in accuracy. Thus reducing the input size does not constitute a reliable approach to speed up the algorithm.

3.1.3 Deploying video-processing in real-world environment

Video processing at the edge has already been integrated into several smart city applications such as smart lights systems (Veena et al., 2016). Deployment at the edge presents many advantages, mainly regarding application responsiveness and data storage and privacy. Edge computing has an important role to play in sustainable data management. For many years it was considered infeasible because of its computational limitations but with the progress of technology, it is now possible to execute advanced processing at the edge of an IoT network.

Carefully designing the data pipeline, where the processing is balanced between cloud and edge computing, is a key element for smart applications and several parameters need to be taken into account. When designing a solution the real-time requirement and the sensitivity of the data should be carefully considered to choose

the right trade-off.

For example, from a crowd monitoring perspective, most CCTV cameras capture only one image per second. When computing crowd density or crowd counting real-time processing (approximately 25 frames per second) is not crucial. Extracting those features every half minute is sufficient. However, other applications like autonomous driving require extreme responsiveness and therefore the computational limitation of the edge device along with network latency is an issue.

Regarding data privacy, video analytics at the edge has many benefits. But the raw images captured by CCTVs potentially contain personal data. However, to perform data-driven decision making, only high-level representations of the data – the density of the crowd or the activity of the crowd – is needed. Computing this information at the edge reduces the cost of using cloud services and ensures better anonymity of the data.

Therefore, how can we improve upon existing deep learning algorithms to make them more efficient in an edge scenario? In the next section, we will review different ways to reduce the computational cost of deep neural networks.

3.2 Reducing computational cost

In their current state, without any adjustment of the model architecture, deploying deep learning applications in production with limited compute power is challenging. For that reason, cloud platforms constitute an ideal choice to perform such heavy compute workloads. But this quickly becomes expensive financially, and despite active effort from cloud providers to create carbon-free data-centres, the environmental footprint of deep learning algorithms keeps increasing and constitutes a major concern (Lacoste et al., 2019) – GPU hardware like the Nvidia V100 GPU or most recent RTX 3080Ti require respectively 300W and 350W power for the graphical processing unit alone. This is not only the case for computer vision algorithms as demonstrated in the previous section, but it also holds true for language models where state-of-the-art model GPT-3 contains up to 175B parameters and requires

several GPUs not only to train, but to run inferences as well just to store the hundreds of gigabytes for the parameters (Bender et al., 2021). Therefore, there is an equal need for pruning on large networks to reduce the size of the models.

To understand how to make deep neural networks more efficient, we first need to understand what causes their high computational cost and what hardware settings contribute most to the current limitations.

3.2.1 Measuring computational cost

As mentioned previously in Chapter 2 Section 2.1.3.1, two factors that cause bottlenecks for running deep algorithms are: (1) the number of mathematical operations – *computational cost*, and (2) the size of the deep network – *memory footprint*. Both are dependant on the neural network architecture and parameterisation, thereby increasing the computational cost and complexity of AI applications. To make deep learning more efficient in order to run inference on low power devices, we need to minimise the use of computational resources by cutting down on the number of mathematical operations, memory usage, or both.

3.2.1.1 Floating-point Operation (FLOP)

Deep neural networks are a succession of compute units organised in multiple layers interacting with one another. In Chapter 2 we described the mathematical background for training or running deep learning models. The computational cost for executing those arithmetic operations can be quantified using FLOPs. Floating-point operations is a measure to quantify the number of any mathematical operations (addition, multiplication, etc.) between two float numbers – float being the default representation for neural networks parameters. Therefore, the speed for executing a deep learning algorithm will depend on how many FLOPs per second the hardware can execute. This information is usually referred to as FLOPS (FLOPs per Second)[1]. This computes power is one of the main distinctions of the performance

[1]FLOPs - floating-point operations — FLOPS - floating-point operations per seconds

between CPUs and GPUs that makes GPUs more suitable to handle deep learning operations, as shown in Figure 3.4.

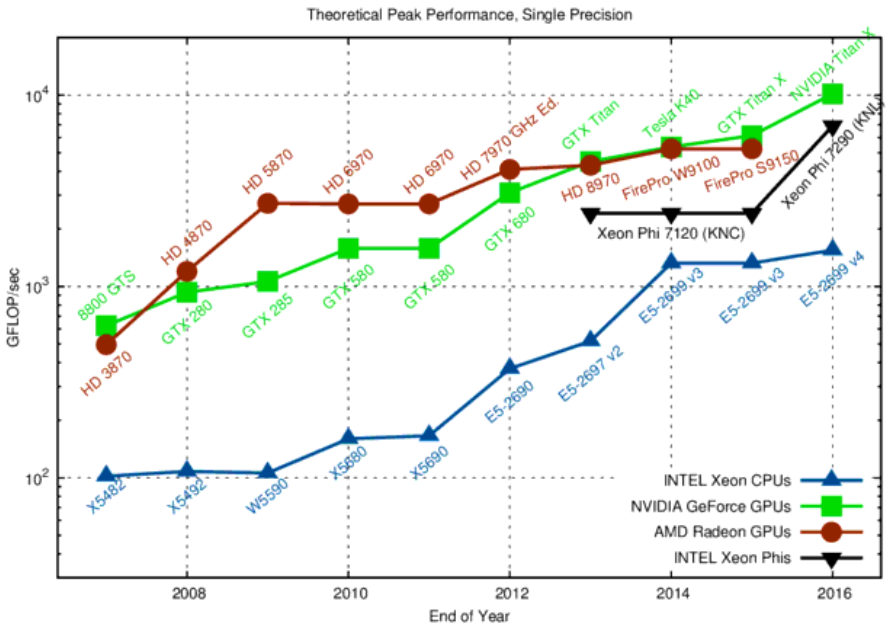


Figure 3.4: Compute performance evolution over the years of GPUs and CPUs measured in GFLOPS. Nowadays there exist even more powerful GPUs with the deep learning standard Nvidia V100 reaching 7 teraFLOPS in double-precision mode. It is a no match compared to CPUs arithmetical compute power. Reproduced from Karl (2013).

The choice of neural network architecture plays a central role in the overall number of FLOPs per model. The number of parameters directly influences the number of mathematical operations but modern architectures often possess custom blocks that may increase the FLOP complexity (e.g., residual blocks, convolution operation, see Chapter 2 Section 2.2.1.2). For instance, to account for local spatiality in computer vision problems, convolutional operations are used. They consist of filters being applied to small patches of the image and slide upon the entire image. Because of this redundancy, they increase the number of FLOPs compared to a simple fully connected layer where the mathematical operations are only computed once.

Note also that the number of FLOPs will differ depending on whether we are training or just running inference on the model. The reason is that when training we have two additional steps to compute the gradients and the update of the parameters,

while in deployment we only care about the forward pass, *aka* the model prediction. However, over-parameterisation not only impacts the number of FLOPs, but also the memory requirement for deep networks.

3.2.1.2 Memory Usage

The memory requirement of deep learning models can be very demanding. During the training phase, we need to store in memory the directed acyclic graph (computational graph) referencing the value of the different parameters in the network and their relationship for the gradient descent steps. In addition, we also need to fit batches of data into the memory for a stochastic optimisation approach (see Chapter 2). For inference, we only need to retain the parameters as the data are usually processed one at a time, and there is no need to compute the gradients to update the parameters.

Although we only need to store the model parameter, current state-the-art models can easily exceed 100 Billion parameters, which is likely to cause issues on low powered or smart devices where memory is limited. Even when the size of the model is more reasonable – around 100 million parameters, it still requires approximately 500MB to store the model. Parameters during training are usually encoded as float32 bits (full-precision) as it encodes a wider range of number resulting in more precise calculus. It is possible to lower the bit precision to reduce the memory requirement and accelerate the compute throughput, for instance one can shift to 16bits or 8bits to gain some storage space during inference without loss in performances (Vanhoucke et al., 2011). However when training lower precision might impair the training, especially during the gradient update as it can cause low gradient values to be zero-out, thus leading to wrong weights update. To overcome this issue, it is possible to use Automatic Mixed Precision (AMP) where 16-bits (half-precision) is used during the forward pass, while 32-bits (full-precision) is kept for the backward pass (Zhao et al., 2019b). Note that some research is even lowering the bit precision further during training time using binary weights during the forward

pass (Courbariaux et al., 2015)

Furthermore, the memory storage requirement can be a bottleneck to maintaining deep learning applications when pushing updates with a modification of the model to the device. When deploying AI applications on remote devices like smartphones, the large model size (MB) can result in high energy consumption, draining the device battery but also consuming excessive internet data to keep the model up-to-date.

3.2.2 Deep learning model compression

Whether for training or inference, over-parameterisation in deep network architectures is a major impediment resulting in high computational requirements. This causes disquietude about the sustainability and accessibility of deep learning technology. High compute hardware is expensive, evolves quickly and reinforces disparities between practitioners who can afford it and those who cannot. This also raises concerns about the life span of specific hardware that quickly becomes obsolete for running new state-of-the-art deep learning models. Finally, this also hinders the deployment of deep learning models in real-world settings in the long run.

Inspired by signal processing research, researchers have thus explored and developed techniques to reduce the parameterisation overhead of deep learning models, trying to compress the model architecture so it can fit within the compute limitations of low-power devices. There are three ways one can compress a neural network architecture, (1) inducing sparsity within the matrix representation of the parameters to remove the less useful parameters; (2) aggregating or clustering parameters together; or (3) implementing more efficient hardware-oriented architectures.

3.2.2.1 Pruning

Because deep neural networks are over-parameterised, a straightforward way to reduce the size of the network is to remove parameters that have little to no impact on the output prediction. This type of compression approach is called *pruning*. Pruning

can be applied in an unstructured way, where parameters are considered individually (Han et al., 2015a), or in a structured way, where parameters are removed in blocks (Liu et al., 2017; Li et al., 2017; Huang & Wang, 2018). Chapter 4 will be dedicated to presenting pruning mechanisms in depth.

By removing parameters from the network we induce sparsity in the structure (matrices with a large proportion of zero values). It is worth noting that if we remove parameters in an unstructured way, we do not necessarily reduce the compute load of the deep learning model because we do not modify the architecture and still need to perform the operation with the zero values and we only reduce the memory storage requirement. On the other hand, structured pruning is more appropriate to reduce the computation as it removes entire parts – filters, learning channels – from the model and can be combined with software optimisation to produce a smaller and faster compact architecture.

3.2.2.2 Quantisation & Factorisation

Another approach to network compression is to reduce the number of parameters by aggregating them together. Quantisation, the action of clustering parameters, can be used to lower the precision of the parameters' encoding. Weights can be aggregated together by low-rank factor compression, where the aim is to reduce the rank of the weight matrices, somewhat similar to what is done in principal component analysis (Denton et al., 2014; Alvarez & Salzmann, 2017). Or one can cluster close weights together (Han et al., 2015a). To some extent, this can be applied to shift the weight representation (real number) to binary `[0;1]` or ternary weight `[0;1;-1]` for even faster inferences (Courbariaux et al., 2015; Alemdar et al., 2017).

Quantisation or *factorisation* model compression directly shrinks the model architecture as opposed to unstructured pruning. It is a very efficient compression approach to reduce compute workload on low-power devices. It can also be used in combination with other compression methods to further reduce the model footprint.

However, this approach is difficult to apply during the training phase.

3.2.2.3 Distillation & NAS

Finally, to compress deep neural networks, one can directly design or employ hardware-oriented architectures through Neural Architecture Search (NAS) or distilling the knowledge of a large network into a smaller, more compact network. NAS can produce a very efficient network. For

instance simple tricks can be used to speed up inferences such as separate convolution operations to reduce latency induced by parameter sharing (Howard et al., 2017, 2019). Nowadays, tools can help find efficient architectures for constrained low-power devices, such as Mobile AutoML (Stamoulis et al., 2020) or MicroNets (Banbury et al., 2021).

Knowledge distillation offers another promising approach to produce more efficient model architectures, especially for NLP models where the number of parameters easily exceeds 1 billion. For such large models trying to compress the original model architecture following other approaches can be challenging. We might not be able to afford to retrain the original model and it is not guaranteed that this will reduce it enough to be efficient. We can leverage those issues by transferring the knowledge of the original network into a smaller more compact network through distillation (Hinton et al., 2015).

3.2.3 On Hardware limitations

Deep learning research is now sufficiently mature for deployment in large scale real-world applications. However, the computational complexity of deep learning models requires them to be optimised if they are to be run on constrained devices. There exists a wide variety of methods to reduce the complexity of deep neural networks, but *is compression enough to run deep learning models efficiently on low-power devices*? In this section we briefly review the state of hardware compute power for AI applications and the progress made over the recent years.

3.2.3.1 DeepLO: Deep Learning Orchestration

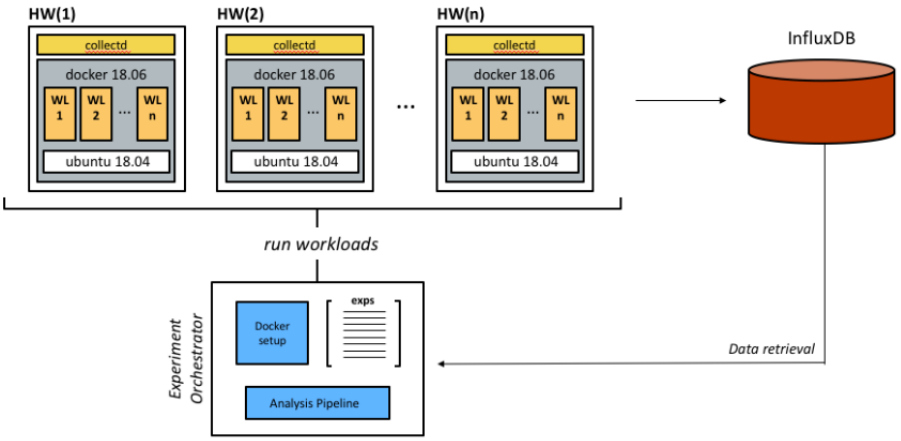


Figure 3.5: DeepLO testbed pipeline. Workloads are deployed on different hardware through docker containers. Different metrics are collected and stored into a database to run deeper analytics off-line.

In a smart environment, many low-powered devices are interconnected together to exchange data and enable monitoring of the environment. When deploying a deep learning application, we need to consider wisely where we place the analytics workload as not all devices are capable of handling it compute-wise. The Deep Learning Orchestration (DeepLO) project run in collaboration with Intel Ireland, is looking at characterising the behaviour of different Deep Learning workloads with different levels of complexity, across heterogeneous Intel Architectures to better anticipate their placement at the edge. Different levels of compression induced by channel pruning (Liu et al., 2017) were tested on two computer vision architectures: a ResNet and a VGG architecture. The different workloads were tested over a wide range of Intel hardware with different CPU compute power – the list of hardware and their characterisation is detailed in Appendix A.2, Table A.2. The main goal in this project was to find useful metrics that operate as key indicators for characterising the workload behaviour and complexity to better anticipate their performance on low-power devices and thus deploy them on the most appropriate hardware. The experimental setup is pictured in Figure 3.5.

For each of the two network architectures, VGG19 (Simonyan & Zisserman, 2015) and Resnet20 (He et al., 2016a), the models were compressed with ratios

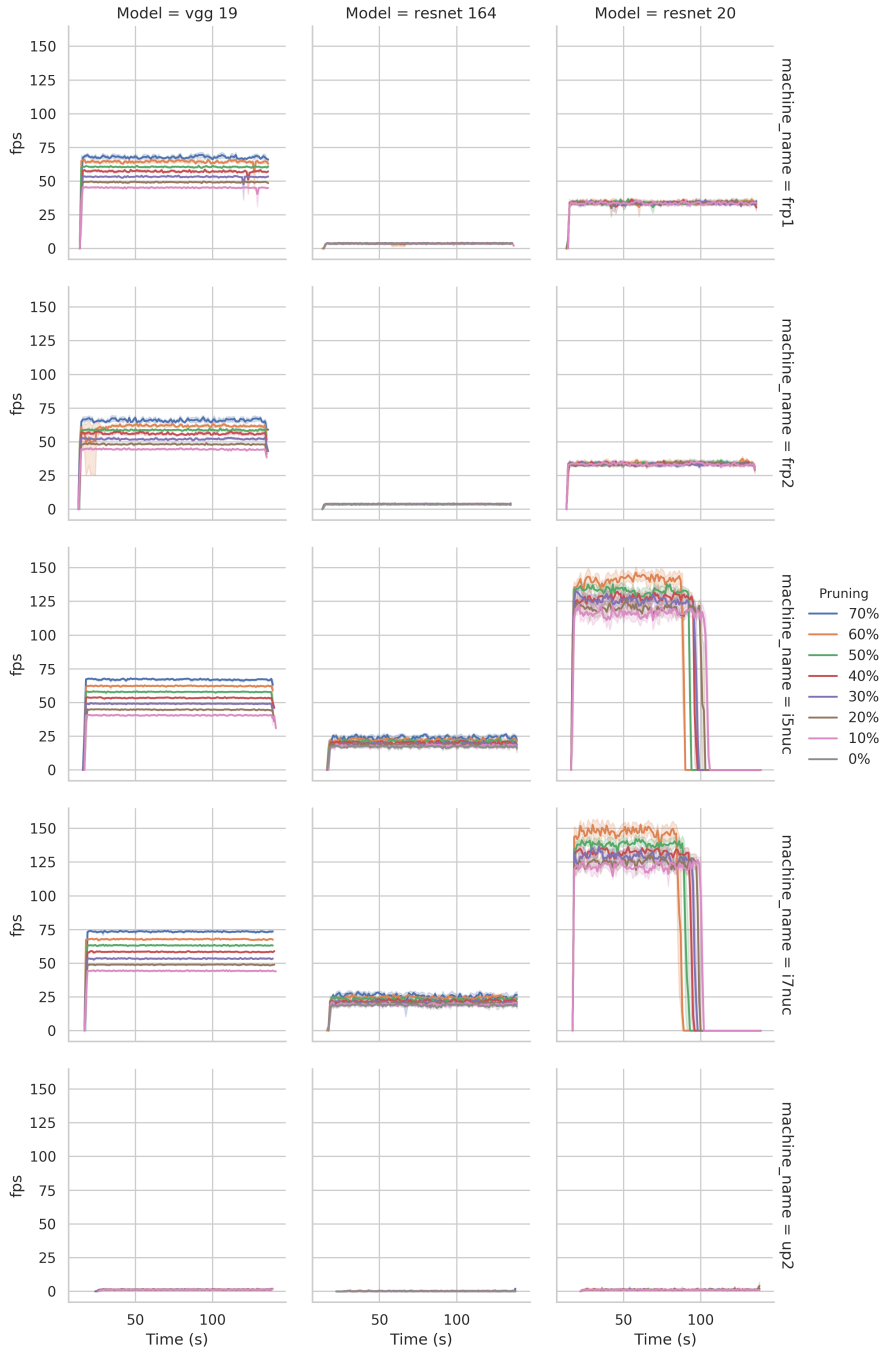


Figure 3.6: Frame per second (fps) ratio for different workloads – levels of compression (colour) – for different model architectures (columns) on the various hardware tested (rows). The behaviour of different workloads is not consistent across different hardware, especially for ResNet-20 workloads. We can observe good fps rates on i5nuc and i7nuc, but on fpr1 and fpr2, the performances are worst than VGG19 while being less computationally demanding (see Table 3.3). up2 hardware is not powerful enough to handle any workload, while ResNet-164 is too complex to be handled efficiently by any hardware. Note that the sudden drop offs observed on the ResNet20 is due to system issues.

from 10% to 70%. In addition, we tested a Resnet-164 architecture to assess the performance of a very deep and complex network. Figure 3.6 displays the frames-per-second processing rate for the different levels of compression (lines color) for different architecture (columns) obtain when running experiments on each of the different devices tested (rows). The experiment consisted on running CIFAR10 image classification task processing one image at a time.

We see that different architectures have very different behaviours. Compression seems to have a much more significant positive impact on VGG19 architectures compared to Resnet20 architectures. This can be explained by VGG19 being far more over-parameterised (39M parameters) than Resnet20 (1.7M parameters), thus easier to compress. VGG19 also obtains a surprisingly better overall FPS rate on certain architectures (frp1 and frp2 – 1st and 2nd row) despite its high number of parameters and FLOPs (see Table 3.3). On very low compute device up2 – last row – all workloads behave equally bad. The typical behaviour we would expect on a hardware device is the one from i5nuc and i7nuc, where the lower the complexity the better the processing performances (fps rate).

To understand what causes such disparate results between VGG19 and Renet20 models on frp1 and frp2, two architectures with a similar number of layers, we need to investigate further hardware metrics. Table 3.3 summarises the complexity of each of the deep learning workloads. Resnet20 architecture is far less complex than VGG19 and thus the difference in behaviour on frp1 and frp2 hardware cannot solely be reduced to a compute-bound, where the algorithms are limited by the compute availability from the CPU. Unlike VGG architectures, Resnet architectures have residual connections. Residual connections were designed to avoid vanishing/exploding gradient problems that arise when training very deep networks. For instance, VGG architectures with greater than 19 layers become challenging to train properly because the gradient signal vanishes and becomes 0, preventing the network from learning. To overcome this, residual blocks implement a skip connection to map the layers in between and allow the gradient to flow properly (see 2,

VGG19				
Pruning Ratio (%)	Best Accuracy (%)	total parameters	Size (MB)	GLOPs
10	92.89	34.3 M	131	0.35
20	92.89	30.3 M	116	0.31
30	92.89	26.9 M	103	0.28
40	92.89	24.1 M	92	0.25
50	92.64	21.8 M	84	0.24
60	84.92	20.6 M	79	0.23
70	10.00	19.6 M	75	0.21
Resnet20				
Pruning Ratio (%)	Best Accuracy (%)	total parameters	Size (MB)	GLOPs
10	93.94	1.5 M	6	0.22
20	93.66	1.3 M	5	0.20
30	93.66	1.1 M	5	0.18
40	93.74	1 M	4	0.16
50	93.90	0.9 M	4	0.14
60	93.46	0.7 M	3	0.12
70	93.38	0.6 M	3	0.10

Table 3.3: Comparison of the properties of different compressed workload for VGG19 and Resnet20 models. Resnet20 is much lighter model to run compare to VGG19. Compression can greatly reduce the number of parameters and FLOPs to help deploy deep learning workload on constrained devices.

Section 2.2.2 for more details). This comes at a low extra compute cost but increased memory requirement as we need to retrieve and connect data from different layers through residual connections. We can hypothesise that Resnet workloads are mostly memory bound as their performance is limited by the memory required to store working data. This could partly explain why Resnet20 models have a slower FPS rate on certain constrained devices where memory management differ.

frp1 and frp2 hardware unlike i5nuc and i7nuc, have a smart cache memory contrary to an l2 cache memory. Intel smart cache technology allocates cache memory dynamically to the core that is the most active, while l2 cache memory divides the cache available equally amongst all cores. As a result, Resnet architectures cannot run efficiently on the different cores as certain cores will be limited in memory.

When monitoring CPU utilisation on a very deep architecture workloads (Resnet164) verses workloads from a smaller architecture (VGG19) – Figure 3.7 – we can notice that the overall CPU usage is in close range, between 90% and 95% except on frp1

hardware which suffers from memory management issue.

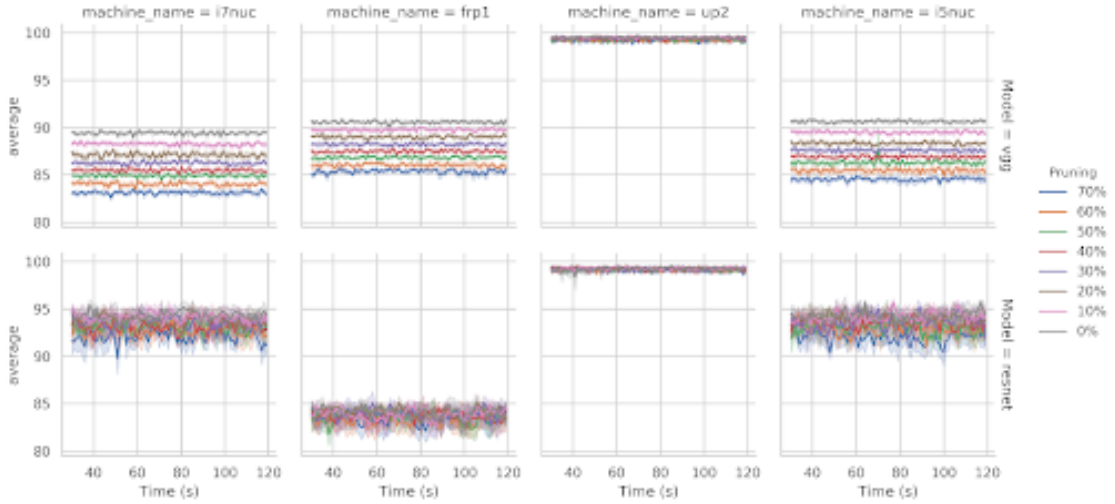


Figure 3.7: CPU utilisation (in %) for different compressed workloads for VGG19 (top row) and Resnet20 (bottom row) on different hardware (columns). We can notice that VGG19 is fairly consistent on all devices whiles Resnet20 fluctuates especially on frp1. up2 does not have the compute power required to handle any of the different workload properly.

Overall, characterising deep learning workload is not as straightforward as it seems and the number of parameters and FLOPs are not sufficient to predict the behaviour on CPU-based devices. Deep learning workload is not only compute-bound but also memory-bound as proven by Resnet workloads. Better monitoring of deep learning workload through robust hardware metrics and improvement on the hardware side (memory cache) is essential to developing more efficient AI applications.

3.2.3.2 Next Gen hardware

The limited arithmetic compute power in CPUs, and the restricted or dynamic memory capacity in low-power devices greatly impedes the deployment of AI applications in IoT settings. Over the recent years, there has been a huge increase in compute power to accommodate training and development of deeper, better performing deep learning models. The deep learning era has accelerated and motivated the research and development of more efficient compute devices like GPUs, and even led to the deployment of more specialised arithmetic compute devices, Tensor Processing Units

(TPUs). However both types of devices require high energy throughput and cooling systems, thus they cannot be deployed on low-power edge devices.

To overcome those limitations, researchers have been looking into the development of specialised hardware to fit *TinyML*, that is, the action of running heavy compute machine learning workloads on tiny low-power devices. Nowadays smartphones are equipped with neural processing units dedicated to running deep learning inference more efficiently. This greatly accelerated the adoption and deployment of deep learning applications. The emergence of new specialised hardware such as the *Intel Movidius* compute stick, Nvidia Jetson, or software libraries such as TensorRT or PytorchLite contributes toward the adoption of AI in our everyday life (see Reuther et al. (2020) for an extensive survey).

3.3 Chapter Summary

Deep Learning offers great potential to interpret and extract knowledge from a wide variety of sensor data. It has proven capable of reaching human-level performance on computer-vision tasks and has great potential applications in smart IoT environments such as smart cities.

In this chapter, we presented a use-case application of deep learning in a real-world environment. We studied how to deploy crowd monitoring analytics in a smart stadium. Traditionally, such workload would be processed using a cloud-centric approach. But the benefits of offering unlimited compute resources comes at the cost of being reliant on a 3rd party and limited by internet connection and bandwidth. Despite being desirable for managing deep learning workloads, it may also increase privacy threats as data needs to transit to a third party, and is also a major drawback for responsiveness.

A middle ground is to balance the analytics between the edge, close to the point of data capture, and the cloud, where deeper long-term analytics can be performed. However, at the edge, compute power may not be sufficient for AI algorithms and running deep learning workloads is highly challenging. As a result, researchers have

developed compression methods to reduce the complexity of deep neural networks and developed specialised hardware for more sustainable and efficient edge processing.

For Deep Learning models to be run on low power devices we need to reduce their complexity. For that, there exists a wide variety of approaches to compress the model architecture and make it less demanding in terms of computational resources. However, compressing the architecture only, may not be sufficient to enable more efficient AI on low-power devices. More efficient memory management is also required.

We presented a comparative study of different deep learning workloads with different levels of complexity and demonstrated that they are not solely compute-bound but also memory-bound. Architectures with specific memory requirements such as Resnet architectures, appear to be very sensitive to the type of cache memory. Dynamic cache memory allocation has the potential to negatively impact deep learning workloads behaviour as the memory between the different cores is not equal, inducing disparities between the processing capabilities.

There has been a lot of progress towards reducing the cost of inference for deep learning models including the development of new hardware and software such as TPUs or Intel Movidius chip. But what limits the compute complexity in deep learning in the first place is the overhead caused by parameterisation. The bigger the model, the better it can learn. However, with this comes an increased requirement for computation. This is not only harmful to inference, but also for training as it creates disparities in access to compute resources and raises concern about the sustainability and carbon footprint of deep learning research.

Inducing sparsity in a neural network through pruning has long been of interest to tackle over-parameterisation. This not only compresses the model for faster inference, but also helps us better understand what part of the network accounts the most for learning, and how can we design more efficient architectures and training methodologies. The remainder of this thesis will examine how to induce sparsity in

a neural network through unstructured pruning. We will see how to design good criterion estimators to determine which parameters to be discarded, and study the impact of heavily sparse networks on the performance.

Chapter 4

The state of sparsity and network pruning

Neural networks are getting bigger, requiring increasing computational resources not only for training, but also for inference. This has significant implications for universal accessibility of the technology with high costs, potential environmental impact of increasing power consumption and inability to use deep learning models on mobile devices and low-power chips.

Pruning consists of reducing the number of parameters in a neural network by removing redundant or low-informative parts while preserving the original network performance. To determine which part of the network can be discarded, the importance of each individual or group of parameters is measured based on some pruning objective. Pruning objectives define how the importance of parameters is measured, it can look at removing the weakest connections – removing parameters with the lowest absolute magnitude, preserving the loss trajectory between the original and pruned model – removing parameters that contribute negatively to the loss using a Taylor approximation, or removing redundant filters by computing similarity score between filters. The ultimate goal is to reduce considerably the compute load while preserving the original network accuracy.

Pruning often constitutes a good entry point for compressing large neural net-

works. It is a simple method that greatly reduces the number of parameters which can be highly beneficial in addition to other compression techniques such as quantisation for better compactness. However, pruning can also be used as a tool to better understand certain training specificity. Do we need over-parameterised models? To what extent does increasing the size lead to better results? Indeed, not only does pruning reduce the complexity of a model, but it can help identify what parts of the network are getting the most attention during training to build better architecture or training procedures.

This chapter presents an overview of the state of sparsity in deep neural networks, with a focus on unstructured pruning applied to computer vision problems. Section 4.1 introduces the core methodology to induce sparsity in a network with pruning, while in Section 4.2 a short history of unstructured sparsity will be presented highlighting major advancement in the field. Finally, Section 4.3 summarises the different frameworks developed over the years compared to one another. This chapter gives the reader a general appreciation for the complexity of pruning research. While the vocabulary present in this chapter was chosen to align with Hoefler et al. (2021) sparsity review paper, the equations and graphs were created by the candidate as a visual aid for presenting the pruning literature.

4.1 Pruning methodology

Pruning can be interpreted as a *mechanism that induces sparsity in a dense network architecture to reduce the computational overhead while maintaining good prediction accuracy.*

In its original state, a deep neural network is at maximum capacity, ie. all the nodes and connections are active. This state is referred to as *dense* network. To reduce the complexity of the original network, parameters can be removed from the architecture or deactivated to lessen memory requirement and compute load. This type of configuration, where lots of parameters have a zero-value, is called *sparse* network in analogy to the mathematical definition of sparse and dense matrices.

Recall that in deep neural networks, parameters values are stored under matrices or tensors. A matrix is said to be sparse when a large majority of its entry have zero-values: (see Equation 4.1).

$$dense = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \end{pmatrix}; \qquad sparse = \begin{pmatrix} 0 & a_1 & 0 & 0 \\ 0 & a_5 & 0 & a_7 \\ a_8 & 0 & 0 & 0 \end{pmatrix} \tag{4.1}$$

4.1.1 Pruning masks

In practice, pruning can be seen as finding and applying a binary mask $\mathbf{m} \in \{0, 1\}$ to the parameters $\boldsymbol{\theta}$ such that $\boldsymbol{\theta} + \Delta\boldsymbol{\theta} = \boldsymbol{\theta} \odot \mathbf{m}$, where $\odot$ is the element-wise product, and $\boldsymbol{\theta}$ the neural network parameters.

The first step is to compute a saliency score for all the parameters considered for pruning based on the chosen pruning objective. Then, parameters are ranked in order of importance and the $k\%$ with the smallest saliencies are masked-out, where k is a hyper-parameters corresponding to the desired compression ratio. To remove the parameters, a threshold corresponding to the k^{th} most salient value is computed, the associated mask can then be generated by masking out – set to 0 – the parameters that fall below the threshold value, whereas the ones above are kept – set to 1. Then the mask is applied to the dense model to obtain the sparse network. Figure 4.1 summarises the different pruning steps.

Pruning almost always results in a loss in accuracy. To recover good performance the sparse network is usually fine-tuned a couple of epochs. The choice of hyper-parameters is crucial during the fine-tuning phase of the pruned network as it can greatly influence the end performance of the pruned model. We will further discuss re-training strategies in Section 4.2.

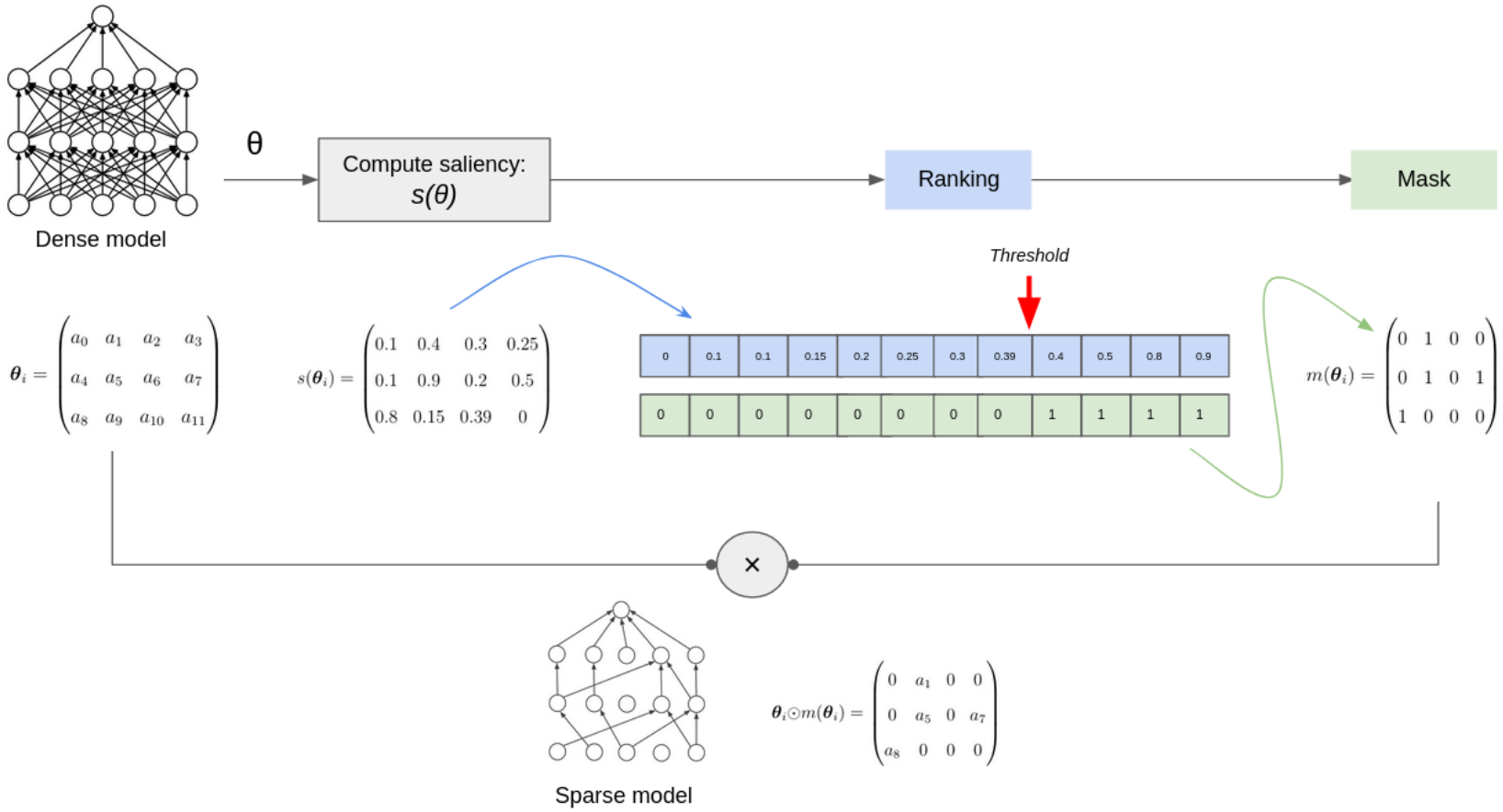


Figure 4.1: Illustration of pruning methodology where $\frac{1}{3}$ of the parameters are removed per layer. $\boldsymbol{\theta}_i$ represent the weight matrice for parameters $\boldsymbol{\theta}$ at layer i. First, the saliency metric is computed, parameters are then ranked by order of importance – in this example the higher the better – and the ones falling below the threshold are removed. A pruning mask is derived from the previous ranking with 0 values for parameters to be discarded and 1 for parameters remaining in the model. An element-wise product between the dense network and the pruning mask is applied to obtain the sparse network.

4.1.2 Pruning criterion and saliency

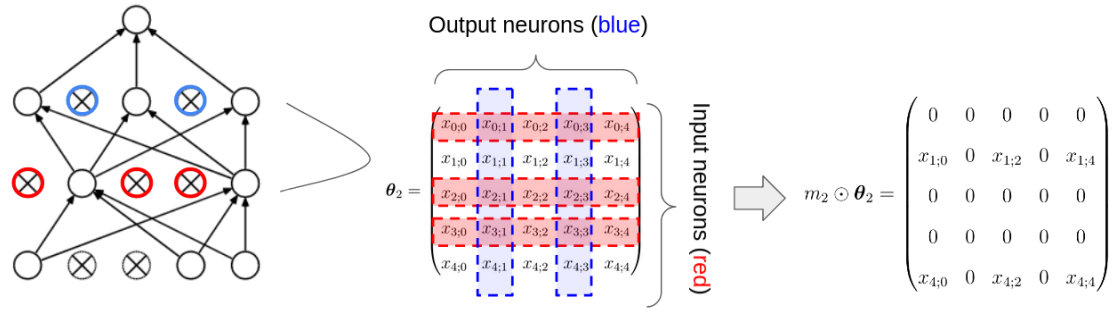
What ultimately determines which parameters are to be discarded, and thus the sparse model performance, is the *saliency* score or *pruning criterion*. A pruning criterion computes an importance measure for each element considered for pruning based on some *pruning objective*. Pruning objectives can be seen as rules or properties to identify unimportant parameters. This can vary from reducing redundancies by computing cosine similarities between filters, removing unimportant learning channels by monitoring the γ parameters of the batch normalisation layers, or simply identifying parameters that contribute poorly to the original network optimisation through loss preservation.

Despite the central role importance measures play in pruning, the performance of sparse models is almost always assessed measuring how far a network can be pruned while preserving good accuracy performance. Only a few little works have tried to investigate pruning criteria further investigating their core similarity Lubana & Dick (2021), or their robustness to adversarial attacks Hooker et al. (2021); Liebenwein et al. (2021), but no work so far has investigated the soundness of pruning objective. *Does better preserving the original network loss or gradient function produces good sparse models?* This question, alongside a detailed presentation of unstructured pruning criteria, will be studied in Chapter 5.

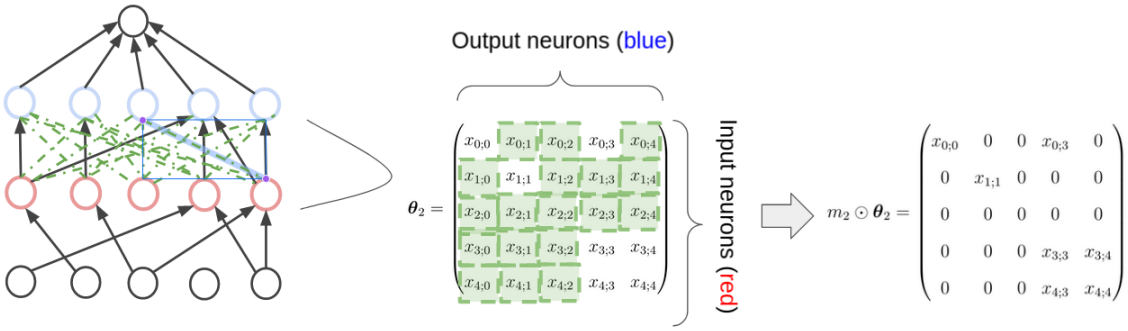
4.1.3 Structured versus Unstructured pruning

Pruning can be applied in a *structured* or *unstructured* way. In the case of a fully-connected network, two type of elements can be removed: the connections, *aka* the *weights* or the *neurons* – biases are ignored in this example. Figure 4.2a pictures the action of pruning neurons, while Figure 4.2b illustrates the action of pruning weights. What differs between both approaches is the element considered for computing the importance measure. Note that parameters belonging to the last layer are not usually considered for pruning as they correspond to the classifier and should not be altered.

When a neuron is removed an entire row or column of the weight matrix, depending on whether the neuron belongs to the input or output layer, is set to zero-values. This type of pruning where parameters are removed in blocks is called *structured* pruning, as sparsity follows a structured pattern. On the other hand, when weights are removed, zero-values are induced sporadically within the weight matrices and no specific sparsity patterns can be observed. This is called *unstructured*.



(a) *Structured pruning*, neurons are removed from the network resulting in row or column being zero-out in the weight matrix.



(b) *Unstructured pruning*, weak connections are removed from the network resulting in associated parameter being zero-out.

Pruning can be applied to a wide range of architectures opening the possibility of elements that can be considered for pruning. For convolutional architectures that include convolution operations or batch normalisation regularisation, structured pruning can be applied to target filters (Li et al., 2017), channels (Liu et al., 2017) or even entire layers could be considered for pruning. Unstructured pruning however always concentrates on removing weak connections from the network – weights – no matter the type of architecture and thus unstructured pruning approaches are more flexible across diverse architectures.

4.1.4 Pruning ratio and scope

In addition to the pruning criteria that defines how to estimate the importance of each parameter, two more hyper-parameters are required to proceed to prune the network: the *ratio* and the *scope* of pruning. *How many parameters should be removed for the pruned network to be efficient*? *Should the same amount of parameters be removed equally across all layers, or should the sparsity per layer differ in order to improve the pruning*?

4.1.4.1 Ratio

A pruning ratio defines how many parameters are removed. It should be high enough to efficiently reduce redundancies amongst parameters, but low enough such that the prediction performance between the original and pruned model is preserved. In summary, picking the right pruning ratio corresponds to finding a trade-off between preserving the original network performance and reducing the computational cost, depending on the pruning motives. The main limitation when defining a pruning ratio is damaging the architecture beyond recovery, this usually happens when one or multiple layers collapse (Tanaka et al., 2020). The point of rupture directly depends on the ratio of parameters remaining in the layer and varies in function of the neural network architecture, the complexity of the task it is trained to solve, and the pruning heuristic (Liebenwein et al., 2021; Hoefler et al., 2021). The more we prune, the more we are at risk of breaking the network architecture. In practice, most pruning ratios lie between 70% and 90%. Also, not pruning enough would result in no gain in memory or compute wise, as the network would remain challenging to run or train efficiently.

Unfortunately, it is not possible to predict the performance of the pruned model prior to fine-tuning the pruned model and there is little research around that matter. Finding the right pruning ratio often requires a greedy exploration. Pruning iteratively – which will be described later in this chapter in Section 4.3 – is a good framework to explore different levels of compression.

4.1.4.2 Scope

Pruning can either be applied at a *layer-wise* level, where the same ratio of parameters is removed from each layer. Or at a *global* level, where $k\%$ of parameters are removed across all layers. The difference occurs when we compute the threshold to determine which parameters are to be discarded, which is done either at the layer level (only saliency scores of parameters from the same layer are considered), or global level (all parameters are considered).

Layer-wise pruning has the advantage of controlling the damage resulting from removing too many parameters from a specific layer of the neural network. As the pruning ratio is applied uniformly across all layers, we ensure that enough parameters are kept to prevent the layers from collapsing. However, this approach is very weak in practice as it lacks the flexibility required to achieve better generalisation and usually result in poor accuracy.

Global pruning is the most common procedure in practice and the one used by default. Despite being more sensitive to high ratios of pruning, with the risk of damaging the architecture beyond recovery, it offers better elasticity to modify and adapt the model architecture. Note that some simple tricks could be enforced to strengthen the robustness of global pruning by enforcing that a minimum percentage of parameters are kept per layer. However, little researches have been conducted to understand the impact of the remaining per-layer sparsity. Recent research indicates that per-layer sparsity has a key role to play in pruning performance for deep networks (Frankle et al., 2021). When pruning is applied before training, the value of the parameters does not matter but rather the ratio of unmasked parameters across the different layers.

4.1.5 Reducing computation

The major difference between structured and unstructured pruning comes from their ability to properly reduce the compute load.

Pruning can either remove low-informative parameters in groups (*structured*

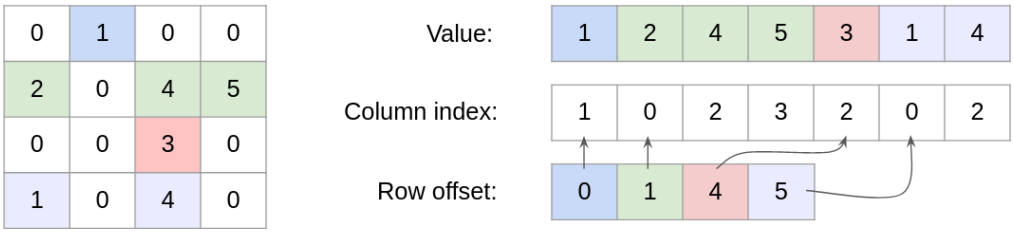


Figure 4.3: Illustration of compressed-sparse-row (CSR) storage requirements to store sparse representation efficiently.

pruning) or individually (*unstructured* pruning). In practice, discarded parameters are replaced by zero-values. If this always reduces storage requirement, it does not necessarily lead to faster computations as the network still need to perform the computations with the zero-values as mentioned before in Chapter 3 Section 3.2.2. When performing pruning in a structured way, however, it is possible to either produce a smaller and more compact dense network – as chunks of parameters can be removed at once (eg. learning channels, neurons, filters) – or use software libraries specialised in sparse calculus to accelerate the computations. This type of approach can significantly reduce the number of computational operations (FLOPs).

By default, Deep Learning models' parameters are stored following a BitMap (BM) scheme, where one bit per parameter is used to store the model parameters. With sparse parameterisation, only the non-zero parameters are required. There exists many tricks to store non-zero parameters: amongst the most common ones we find Compressed-Sparse-Row (CSR) format (see Figure 4.3). It consists of holding a row-pointer pointing onto any non-zero values and its associated column index in form of arrays. In that configuration, 3 bits are required to represent one element. The dense network needs to be compressed by a minimum factor of 3 if we want to observe any storage benefit as there is a 1-2$\times$ overhead in storage for non-zero values. In practice, this means a certain level of sparsity needs to be reached as more bits per element is required for storage, usually high levels of compression around $\approx 90\%$. Other methods include Compressed-Sparse-Column (CSC) – the equivalent of CSR holding column pointers, or Coordinate Offset (COO) – instead of pointers

vector it simply holds the column and row index for every non-zero value.

If unstructured pruning doesn't provide any direct gain in computation, recent research tends to show that wide and sparse networks are better at learning and generalising than their smaller dense counterpart produced by structured pruning (Li et al., 2020; Golubeva et al., 2020). Moreover, unstructured pruning is a more flexible approach that can reach a higher compression ratio in practice without suffering a significant loss in accuracy. The field of sparsity in deep neural networks is recent and in the coming years, hardware and software are more likely to be improved to enable faster computation for sparse networks.

4.2 A short history of unstructured pruning

4.2.1 A way to reduce over-parameterisation

Early pruning work in deep learning can be dated as far as the late 90s (Hassibi & Stork, 1992; LeCun et al., 1989). At that time, deep learning research was still in its early stages and real-world applications were fairly limited. However, there were already concerns about the complexity and size of deep learning models expected to increase in the future, causing slower training and poor generalisation due to the large amount of over-parameterisation. Both methods developed at that time, Optimal Brain Surgeon (Hassibi & Stork, 1992) and Optimal Brain Damage (LeCun et al., 1989), proposed to estimate the importance of parameters based on a quadratic model of the loss and remove those that have the least influence on the loss. Those methods will be presented in further detail in Chapter 5.

Despite promising results, the high computational costs for computing quadratic approximations of the loss and the emergence of strong regularisation techniques to tackle over-parameterisation – such as Dropout and Batch Normalisation described in Chapter 2 – dismissed the need for such pruning approaches. Two decades later interest in pruning was renewed with the desire to run deep learning applications on low-power devices. After the ImageNet revolution in 2012 (Krizhevsky et al.,

2012), and the rise of GPU-oriented training, training large models became easier than ever and was followed by a quick jump in model complexity (see Chapter 1, Figure 1.1). Deploying AI applications in production became problematic due to computational constraints with the major drawback being the large number of parameters in modern deep learning models, thus the need for smaller models.

4.2.2 Deploying AI in productions

In 2015, Han et al. (2015a) proposed one of the first successful compression approach to reduce significantly the size of deep convolutional neural networks and lessen the computational cost to improve their efficiency for low-power devices. Their work relied on a three stages compression pipeline involving *pruning*, to reduce the number of weights, *quantisation*, to aggregate weights together and reduce their bits representation, and an *Huffman encoding* on top of that to further reduce computations. With their method, Han et al. (2015a) were able to reduce by $\times 49$ the size of a VGG16 model trained on ImageNet with only a 0.33% loss in top-1 accuracy.

Their work inspired many others to explore pruning as a simple tool to compress deep neural networks with the emergence of structured pruning (He et al., 2017; Liu et al., 2017) amongst different strategies to apply pruning.

4.2.3 Better understanding deep learning training

A second major breakthrough for pruning research happened in 2018 when an interesting property of deep neural networks was discovered thanks to pruning research. For a long time, it was believed that dense over-parameterised models were required in order to train high performing networks, especially at initialisation to reinforce optimisation procedures. Indeed, very sparse networks by definition could not learn from scratch and therefore pruning could only be applied during or after training. However, those two concurrent works showed that sparse neural networks could actually be trained from scratch.

Liu et al. (2019) demonstrated that it was possible to re-train a pruned network

from initialisation without losing accuracy. They compared the performance of a pruned network after fine-tuning and when retrained from initial weights values. They showed that for structured pruning, training from scratch was able to achieve better results than fine-tuning for a wide range of models and datasets (CIFAR10, CIFAR100 and ImageNet). For unstructured pruning though, pruned networks obtained through one iteration of pruning (*oneshot* pruning) succeed to train from scratch on smaller datasets (CIFAR10, CIFAR100) but failed to scale on larger datasets (ImageNet).

In parallel, Frankle & Carbin (2018) proposed the following lottery ticket hypothesis: *within a deep neural network, there exists a sub-network (winning lottery ticket) that when trained in isolation can reach similar performance than its original dense counterpart.* To find such winning lottery tickets, the authors proposed to cyclic pruning framework that iteratively prunes a small portion of the weights through magnitude pruning heuristic, and retrains the subsequent sparse model from scratch until the desired trade-off – sparsity/performance accuracy – is met. While their methods worked well on small scale datasets (CIFAR10, CIFAR100), it failed to scale to more complex tasks (ImageNet). To overcome these limitations, the same authors proposed to rewind the weights to a later epoch – where the sub-network is more stable to SGD noise – rather than retrain from initialisation and were successful at training a very sparse network from early training (Frankle et al., 2019).

Those two works highlighted an important property of sparse networks: it is possible to train very sparse networks from scratch more efficiently than their dense counterpart with faster convergence rate and better generalisation. This opened up new perspectives on training neural networks and attracted a lot of new researchers to the field eager to better understand and explore deep learning training dynamics.

For a long time if was believed that reducing over-parametrisation would lead to more robust models better at generalisation, however researcher Sara Hooker (Hooker et al., 2020, 2021) demonstrated that it was quite the opposite. By removing pa-

rameters from the original dense model, pruning was actually reinforcing existing bias in the training data, producing models weaker towards adversarial attacks and less fair. This is an important observation for the limitation of pruning and the hidden benefits of over-parameterisation. If her research only studied the case of magnitude pruning approach, in Chapter 6.2 we will explore fairness and pruning for other unstructured pruning methods.

4.2.4 Complexity of pruning research

Network pruning has proven to be a simple method, easy to implement that can produce very performing networks with a much lower computation cost. But it can also be a great method to help understand training dynamics in order to build performing models with a low compute cost and better generalisation. Because of this duality, pruning research lies in between *application-oriented* and *empirical theoretical* research, where goals and expectations are not the same.

Whether the model is pruned at initialisation or during/after training, pruning is always followed by a retraining phase. This step is crucial as it enables the pruned network to recover from potential loss in accuracy and re-adjust its connections and learning representations. Originally, fine-tuning was held such that we would retrain the sparse model from the point of pruning keeping the same hyper-parameters. But recent researches have shown that hyper-parameters play a crucial role in the pruned network performance, especially the learning rate (Renda et al., 2020; Le & Hua, 2021). In Chapter 6 we will expand upon the importance of more careful fine-tuning of the pruned model.

Other researchers motivated by the theoretical aspect of pruning have attempted to better understand pruning masks, to build stronger sparsity metrics but discovered that masks are quite flexible and it is not yet quite clear what makes a good pruning mask (Zhou et al., 2019; Paganini & Forde, 2020; Frankle et al., 2021). But the theoretical aspect of sparse network also explore sparse optimisation (Evci et al., 2020b,a; ab Tessera et al., 2021), training with sparsity in the data (Paul

et al., 2021), neural architecture search (Stamoulis et al., 2020), etc.

On the other hand, application-oriented research seeks smaller computational footprints, faster inference, more compactness. Those will be mainly centred around structured pruning and software optimisation. But compression research lack a clear benchmark and it is hard to compare pruning methods together (Blalock et al., 2020).

4.3 Pruning Frameworks

There exists a wide variety of ways one can induce sparsity in a deep neural network as briefly mentioned before. This section will review different pruning frameworks developed over the years around unstructured pruning. Note that *dynamic pruning* and *ephemeral pruning* are outside the scope of this study and will not be reviewed. The reader is invited to check Hoefler et al. (2021) for more information.

4.3.1 One-shot

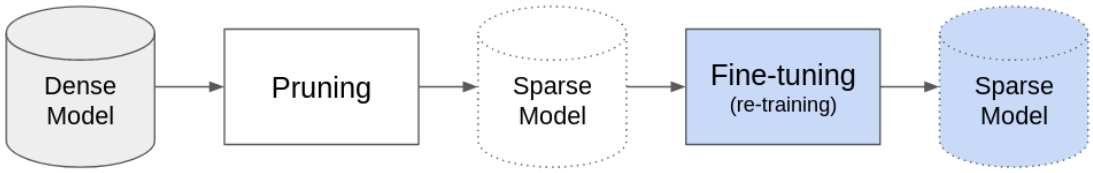


Figure 4.4: *One-shot* pruning. A straightforward way of removing parameters where the model is pruned and train only once.

One-shot pruning is the action of removing parameters in one pass: the pruning and fine-tuning of the pruned model is performed one time, hence the name *one-shot* pruning (see Figure 4.4). This method has the advantage of requiring minimum extra computations as the pruning cycle is only performed once. Thus, there is no overhead in computing the importance measure multiple times or retraining the pruned model. It can be applied any time during or before training but it is preferable to apply it on a partially or fully trained model.

When working with a high ratio of pruning if a too large portion of parameters is removed at once, it might hinder the overall performance of the pruning heuristic as we are more at risk of damaging the architecture. It might result in important parts of the network – an entire layer or important connections – being deleted that would not be otherwise. The architecture has a higher chance of being damaged beyond the point of recovery. For this reason, this type of pruning framework is not often used in practice when working with a high compression ratio. One-shot pruning is most favoured in structured pruning when working with a fully trained network and trying to reduce redundancies across filters for instance, but it has few practical uses in unstructured pruning approaches.

4.3.2 Iteratively

Pruning large chunks of parameters at once carry the risk of damaging the network architecture beyond recovery. A connection can be of low importance in presence of neighbouring parameters, but when all are removed, the remaining connection might become crucial for the prediction score of the pruned model. By using a high ratio of pruning in a one-shot setting, there is a possibility that those important connections are missed and removed from the neural network. A straightforward solution to it is to remove parameters in small chunks multiple times, instead of once. This is called an *iterative pruning* framework.

Parameters can be removed progressively throughout training as illustrated in Figure 5. When using this type of approach, two hyper-parameters are introduced: the frequency of pruning (n) and a pruning ratio scheme. The frequency of pruning corresponds to the different epochs at which we should perform pruning. While the ratio scheme will determine how many parameters to be removed during each pruning iteration. Two common schemes are linear pruning, where the same number of parameters is removed at each iteration, or exponential pruning, where fewer and fewer parameters are removed. Exponential pruning is usually preferred in practice as the fewer parameters remaining in the network, the more careful one should

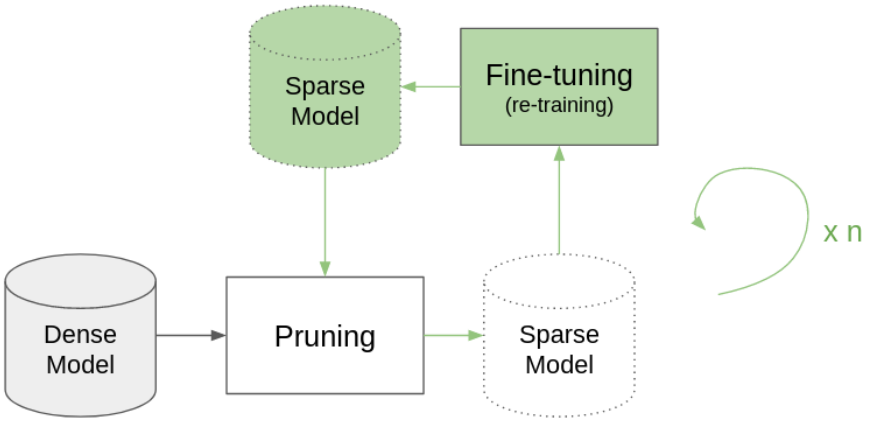


Figure 4.5: *Iterative* pruning. The cycle of pruning-retraining is performed n times to iteratively remove small chunks of parameters at a time. By performing the pruning cycle multiple time, the pruning heuristic adjust itself and we are able to better preserve the network architecture.

select which parameters are to be discarded as the removal of a parameter can highly impact the values of others neighbouring parameters (Chapter 5 will present different pruning schedules in more details).

Iterative pruning prevent overhead in fine-tuning, as retraining of the pruned model is done in parallel of pruning. It can provide a safety net when exploring high pruning ratios as it offers the possibility to roll back to a previous pruning instance to find the optimal trade-off between compression and performance accuracy. But it requires estimating the importance of parameters multiple times. When simple criteria that consist of removing the lowest weights magnitude are used to estimate the importance of a parameter there are no significant additional costs, but for more complex heuristics it can come at a higher computational cost.

For all those reasons – fine-grained selection of parameters, and safety-net to high pruning ratio – iterative pruning is the most common one used framework in practice (Zhu & Gupta, 2017; Frankle & Carbin, 2018).

4.3.2.1 Lottery Ticket Hypothesis

Iterative pruning framework is at the core foundation of the Lottery Ticket Hypothesis briefly mentioned earlier (Frankle & Carbin, 2018). Instead of continuing to train the sparse network from the point of pruning, the parameters are set back to

their initialisation values and fine-tuned of the pruned network is done from scratch (see Figure 4.6). By resetting the weights to their original values, or later in training – rewinding (Frankle et al., 2019), the pruning heuristic is able to better estimate the importance of parameters as the network can fully recover through a full cycle of training. This framework was originally developed with computer vision models (CNNs) using a magnitude heuristic – removing the weights with the lowest magnitude $|\boldsymbol{\theta}|$), but this framework has proven to be very flexible to other domain (Morcos et al., 2019) including natural language processing (Chen et al., 2020), reinforcement learning (Yu et al., 2020), or even semi-supervised and unsupervised learning (Chen et al., 2021).

A major drawback of the iterative approach is that it is not cost-efficient. The network needs to be re-trained from scratch multiple times before obtaining a good sub-network sparse enough to lower the computational load. For instance, 15 iterations of lottery tickets iterations are required to prune the model up to 95.6%, following an exponential pruning scheme with 20% removed each time as described in the original paper, meaning the network has to go through 15 training/pruning cycles. However, the lottery ticket framework constitutes a good baseline for better understanding training and learning schemes of deep neural networks (Frankle et al., 2021; Liebenwein et al., 2021).

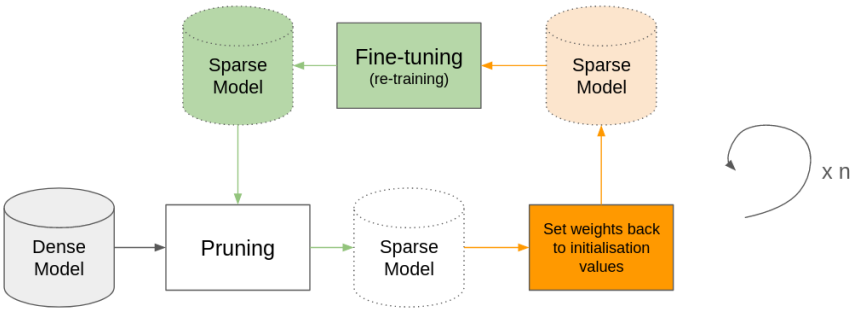


Figure 4.6: *Lottery Ticket* framework. Similar to iterative pruning presented in Figure 4.5, pruning is perform multiple times at the exception that weights are set back to their values at initialisation prior to fine-tuning, training the sparse network from scratch each pruning cycle.

4.3.3 Multi-stage

To alleviate the extra computation induced by iterative pruning, pruning can be performed multiple times without re-training the model in between. With a multi-stage pruning approach, the importance measure is adjusted to the state of the neural network and we can remove small chunks of parameters and obtain a fine-grained pruning mask with only one pass of fine-tuning (see Figure 4.7).

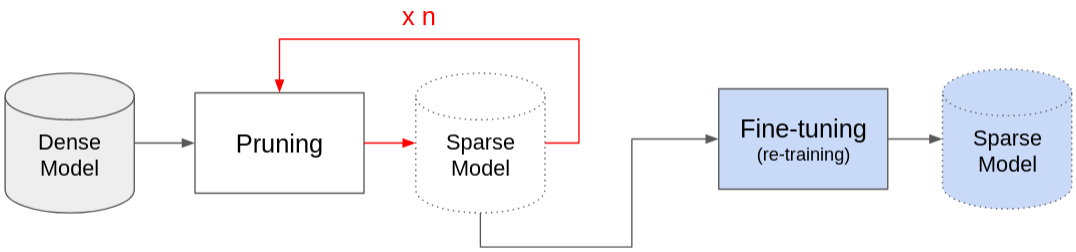


Figure 4.7: *Multi-stage* pruning. Similar to iterative pruning, the pruning is performed multiple times but we do not retrain the sparse model in-between each pruning steps. This lower the computations while removing parameters little at a time.

Magnitude pruning importance measure removes the least salient connections deduced from the parameters absolute value $|\boldsymbol{\theta}|$. To adjust $|\boldsymbol{\theta}|$ heuristic after removing a small portion of the parameters, the pruned model need to be retrained otherwise the magnitude of individual parameters will remain the same. However, some other unstructured pruning criteria based on loss or gradient approximation of the dense neural network can be updated. For these, it is possible to re-estimate the pruning heuristic *without* retraining the model by recomputing the loss or gradient approximation with the sparse model prior to re-training its weights. This produce more reliable saliency scores and better preserve the network architecture.

The number of pruning stages that one should perform is not properly defined. In theory, the more the pruning iterations, the better the pruned model should be. In the literature, researchers usually perform 100 iterations of pruning but this number is often overlooked (Tanaka et al., 2020). Moreover, some pruning criteria can be expensive to compute and despite reducing the need for multiple iterations of training - fine-tuning, this might still hinder the overall compute time

spent pruning. The impact of performing multiple stages of pruning will be further discussed in Chapter 5.

4.4 Chapter Summary

This chapter presented an overview of the state of sparsity and network pruning. In Section 4.1 the core methodology for inducing sparsity in a deep neural network was introduced. Pruning consists of removing the least salient parameters from the network while preserving the original (dense) network accuracy. The saliency of a parameter is computed through an importance measure, also called pruning criterion, derived from some pruning objective that determines what properties indicates the usefulness of a parameter. Parameters to be discarded are then set to zero and the resulting mapping is called a pruning mask. After obtaining a sparse model, a fine-tuning or re-training is required for the model to recover any loss in performance. Almost any type of parameters can be removed from the network, when parameters are removed in chunks (neurons, filters, channel) we call it *structured* pruning, on the other hand removing individual parameters (weights) is called *unstructured* pruning. While structured pruning offers a greater reduction in terms of computing operation, unstructured pruning is more flexible and can be applied to any type of architecture.

Following this introduction to sparse networks, Section 4.2 presented a short history of pruning research, highlighting major advancements in the field that lead pruning to expand to other research interests, outside solely creating more compute efficient models. Over years of research, pruning shifted from application-oriented motives (lower computation costs, better generalisation, acceleration on low-power devices), to consideration towards an empirical and theoretical understanding of deep neural networks (sparse training, dynamic training, role of parameterisation) through unstructured pruning. Nowadays the sparsity community is scattered around between academia and industry with disparate problems often sharing the same interest in network sparsity. Despite a growing community large

enough, there is no common ground (lack of benchmark) or venue (audience) to witness and share cross-cutting research problematics.

In this thesis, we focus on unstructured pruning approaches because of their flexibility and great potential to help design better training schemes. A variety of unstructured pruning frameworks were developed presented in Section 4.3. While iterative pruning approaches prevent pruning from damaging the network architecture beyond recovery contrary to one-shot pruning, it comes at the cost of multiple pruning-retraining cycles that can quickly become a burden computationally. This is a major computational drawback that practitioners cannot always afford to spend in practice. Multi-stage pruning mimics an iterative pruning approach removing the need for a fine-tuning phase between each iteration of pruning.

Globally there is a lack of understanding of what causes good performances when pruning a deep neural network. Many factors can influence the performance of the pruned model: the choice of pruning criteria, the pruning framework, epoch, ratio, the retraining strategy, the hyper-parameters etc. Only a little research has been done into understanding the influence of the learning rate (Le & Hua, 2021) or pruning epoch (Frankle et al., 2020) but none has assessed the very foundation of pruning criteria. The pruning criterion constitutes the first step toward building a sparse architecture. To build more efficient pruning strategies, we need to understand what in existing criteria helps build more robust pruned models.

In Chapter 5 we study the integrity of pruning criteria. We assess whether best preserving the original, dense network functions and properties is a good strategy to ensure good accuracy performance of the pruned model. In Chapter 6 we evaluate the impact of parameterisation and sparse architecture beyond solely looking to the accuracy performance in order to better understand the impact of inducing sparsity in a deep neural network.

Chapter 5

Pruning Metrics

The goal of unstructured pruning is to remove weights from the network such that the parameters vector representation after pruning is of desired sparsity $\kappa \in [0, 1]$. To do so, pruning criteria are designed to find a step $\Delta\boldsymbol{\theta}$ to add to the current parameters $\boldsymbol{\theta}$, where $\boldsymbol{\theta} \in \mathbb{R}^D$, such that the ratio of parameters remaining after the pruning step $\|\boldsymbol{\theta}+\Delta\boldsymbol{\theta}\|_0$ is equivalent to $1-\kappa$. This can be interpreted as finding and applying a binary mask $\mathbf{m} \in \{0, 1\}$ to the parameters such that $\boldsymbol{\theta} + \Delta\boldsymbol{\theta} = \boldsymbol{\theta} \odot \mathbf{m}$, where $\odot$ is the element-wise product. While doing so, the performance accuracy between the original and pruned network should be maintained.

To find $\mathbf{m}$, pruning criteria compute a *saliency* score for each parameter based on some *pruning objective* that aims at preserving or improving certain properties of the original network. For instance, one could wish to preserve the loss function to maintain the training trajectory, or improve the signal propagation to ensure good re-trainability of the pruned network. Parameters can then be ranked in order of importance, and the ones with the smallest saliency scores are pruned – set to zero (ie masked out from the network) – while the ones with the largest saliency are kept unchanged.

In early pruning work, Optimal Brain Damage (OBD) (LeCun et al., 1989) and later Optimal Brain Surgeon (OBS) (Hassibi & Stork, 1992), the importance of each parameter was estimated by approximating the effect of removing it on the loss function. This type of approach focuses on preserving the loss function and

is known as *loss-preservation* criteria. Over the years, various heuristics have been developed to prune the model more efficiently using different pruning objectives, in addition to *loss-preservation* approaches (Lee et al., 2019; Molchanov et al., 2017, 2019), new criteria based on preserving parameters with high *magnitude* (Han et al., 2015b,a; Zhu & Gupta, 2017), or looking at preserving the signal propagation of the network through gradient information (Wang et al., 2020; Tanaka et al., 2020), have been designed.

Each pruning criterion will remove a different set of parameters from the network. This chapter is dedicated to reviewing the different unstructured pruning criteria – *magnitude*, *loss*, and *signal* preservation –, and study the soundness and theoretical limitation behind the design of each criterion family. *How to select which parameters to be discarded*? *How to improve current pruning objective and criteria*? *Does preserving the original network function actually lead to better performing pruned model*? These questions are an overview of the questions that will be investigated throughout this chapter.

If we were to design better unstructured pruning strategy, we first need to understand to which extent are existing pruning criteria good at producing efficient pruned networks. To identify what characteristics of an importance measure leads to good performing pruned models, we review and compare the major unstructured pruning criteria designed over the years. First, all the different unstructured pruning criteria developed over the years are presented and unified under a same umbrella and split into three families: (i) magnitude-based, (ii) loss-preservation and (iii) gradient-flow. Then, in a second time an empirical evaluation is conducted to assess the integrity of the different approaches listed. Note that the work that appears in this chapter is the candidate alone or joint work with colleagues at MILA (Université de Montréal) that resulted in the following publication (pre-processing) Laurent et al. (2020).

5.1 Pruning criteria families

In unstructured pruning, there exists three main families of pruning criteria that implement different pruning objectives: (i) *magnitude-based*, (ii) *loss-modelling* and (iii) *gradient-based* criteria. The following section presents and describes the theoretical foundation behind each family.

Throughout this chapter, importance measures used to determine which parameters to discard will be defined by s for saliency. Pruning criteria will be referred to by their acronyms defined in the header of each criterion section. For the reader's clarity, note that *model* can refer to either the neural network architecture (eg. *pruned model*), or the mathematical model used for computing the pruning saliency (eg. *pruning model* referring to the loss model or *underlying model*). In most cases, model will refer to the second one except if stated otherwise.

5.1.1 Magnitude pruning

5.1.1.1 Pruning Objective

Magnitude pruning objective consists of removing the least salient parameters, or connections, based on their absolute value. Connections in a neural network are represented by weights are defined for a given layer l by the weight tensor $\boldsymbol{\theta}_l$. The sensitivity of the signal flowing through the neural network is modified by the activation units but also by the strength of the connections associated with it (see Chapter 2, Section 2.1). Weak connections are synonyms of low signal intensity and usually result in little information carried by these parts of the network, thus they can be removed. When applied in an unstructured way, *magnitude pruning* consists of removing the least salient connections (individual weights), or route, from the network. Note that original pruning methods used outside deep learning, in graphs theory or decision trees, generally employ magnitude as a pruning method.

5.1.1.2 Magnitude Pruning (MP)

MP is a popular pruning criterion in which the *saliency* is simply based on the norm of the parameter:

$$s_k^{\text{MP}} = \boldsymbol{\theta}_k^2 \tag{5.1}$$

The idea of removing parameters from the network was first introduced as a regularisation tool to overcome over-parametrisation in fully-connected layers (Dropout (Srivastava et al., 2014), Dropconnect (Wan et al., 2013)). Inspired from the decision tree literature, parameters were randomly dropped during each training iteration to reduce co-adaptation between neural units leading to more robust models. However, when convolutions layers were introduced (He et al., 2016a) dropout-like methods showed their limitations and were not a strong enough regulariser anymore (see Chapter 2, Section 2.2.2. This led to the creation of Batch Normalisation (Ioffe & Szegedy, 2015), a much stronger regulariser that normalises shifts induced by training in small batches of data and is robust under spacial filtering operations. It is only in 2015 that Han et al. (2015b) re-introduced the concept of dropping low informative parameters in the context of pruning under magnitude pruning.

Nowadays, despite its simplicity, MP has proven to work extremely well in practice (Han et al., 2015a; Gale et al., 2019), and is used in current state-of-the-art frameworks (Renda et al., 2020; Frankle et al., 2021). As a result, MP constitutes a strong baseline for comparison in pruning criteria literature.

5.1.2 Loss-modelling

5.1.2.1 Pruning Objective

Pruning is almost always followed by a fine-tuning phase. To preserve the network performance after pruning, loss-modelling criteria aim at preserving the network

trajectory dynamics by minimising the change in loss function between the pruned and original model (Lubana & Dick, 2021) to ensure good retrainability of the pruned model. In other words, Loss-modelling criteria try to find a step $\Delta\boldsymbol{\theta}$ such that the loss in the pruned model $\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})$ does not differ significantly from the one of the original model $\mathcal{L}(\boldsymbol{\theta})$, leading to the following pruning objective:

$$\underset{\Delta\boldsymbol{\theta}}{\text{minimize}} \quad \Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) \stackrel{\text{def}}{=} |\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta})| \tag{5.2}$$

Directly solving this problem would require evaluating $\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})$ for all possible values of $\Delta\boldsymbol{\theta}$ – ie. estimating the impact of removing every single parameter on the loss individually, and removing the one that changes the loss the least – which is prohibitively expensive for modern architectures where the number of parameters easily surpass 100 million. To leverage this issue, the loss function is approximated, usually using Taylor expansion, to rely on heuristics to find good solutions. In practice, common loss-modelling pruning criteria use either the first or second order of the Taylor expansion to model the loss and estimate which parameters can be discarded. For instance, early pruning work, Optimal Brain Damage (OBD) (LeCun et al., 1989) and later Optimal Brain Surgeon (OBS) (Hassibi & Stork, 1992), proposed to estimate the importance of each parameter by approximating the effect of removing it, using the second-order term of a Taylor expansion of the loss function. Later, newer methods based solely on the first-order term were developed (Lee et al., 2019) to tackle pruning at initialisation. The authors stipulated that preserving the trajectory minimising the change over the loss function was a more robust approach when pruning at initialisation, in comparison to their magnitude pruning counterpart, as parameters are randomly initialised following a Gaussian distribution.

In summary, loss-modelling can be approximate using first order, second-order methods, or following quadratic models that include both first and second orders.

5.1.2.2 Optimal Brain Damage (OBD)

Second orders loss approximation methods were the very first type of pruning criteria approaches developed (LeCun et al., 1989; Hassibi & Stork, 1992). Back in the 1990s, Deep Learning research was in its early stages and there was no need to deploy large models in real-world production, thus compressing the model. However, before the wide adoption of regularisation methods, it was believed that over-parametrisation in deep networks was harming the model, preventing it to learn properly. Hence, researchers tried to remove unnecessary parameters evaluating their influence on the training through the loss.

LeCun et al. (1989) proposes to approximate the loss $\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})$ with a quadratic model leading to the following approximation of $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$:

$$\Delta\mathcal{L}^{QM}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = \left| \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}^{\top} \Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^{\top}\mathbf{H}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \right| \tag{5.3}$$

where $\mathbf{H}(\boldsymbol{\theta})$ is the Hessian of $\mathcal{L}(\boldsymbol{\theta})$. Having to compute the hessian is an obvious drawback of quadratic models, as it is intractable even for small neural networks. It is common to approximate $\mathbf{H}(\boldsymbol{\theta})$ using the Generalised Gauss-Newton $\mathbf{G}(\boldsymbol{\theta})$ approximation (Schraudolph, 2002):

$$\mathbf{H}(\boldsymbol{\theta}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \frac{\partial f_{\boldsymbol{\theta}}(x_i)}{\partial\boldsymbol{\theta}}^{\top} \nabla^2_{u=f_{\boldsymbol{\theta}}(x_i)} \ell(u, t_i) \frac{\partial f_{\boldsymbol{\theta}}(x_i)}{\partial\boldsymbol{\theta}}}_{\mathbf{G}(\boldsymbol{\theta}),\ \text{the Generalized Gauss-Newton}} + \underbrace{\sum_{k}^{K} \frac{\partial\ell(u, t_i)}{\partial u_k}\Big|_{u=f_{\boldsymbol{\theta}}(x_i)} \frac{\partial^2 f_{\boldsymbol{\theta}}(x_i)_k}{\partial\boldsymbol{\theta}^2}}_{\approx 0} \tag{5.4}$$

$$\approx \mathbf{G}(\boldsymbol{\theta}) \tag{5.5}$$

where K is the number of outputs of the network. $\mathbf{G}(\boldsymbol{\theta})$ has the advantage of being easier to compute and is positive semi-definite by construction. Although LeCun et al. (1989) uses $\mathbf{H}(\boldsymbol{\theta})$ in the equations of OBD, it is actually $\mathbf{G}(\boldsymbol{\theta})$ which was used

in practice (LeCun, 2007).

OBD make two more assumptions are made: first, it assumes the training of the network has converged, thus the gradient of the loss wrt $\boldsymbol{\theta}$ is 0, which makes the linear term – $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \Delta\boldsymbol{\theta}$ – vanish. Second, the interactions between parameters are neglected, which corresponds to a diagonal approximation of $\mathbf{G}(\boldsymbol{\theta})$, leading to the following importance measure to quantify the influence of individual parameters on the loss:

$$\Delta\mathcal{L}^{OBD}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}_k) \approx \frac{1}{2}\mathbf{G}_{kk}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}_k^2 \qquad \Rightarrow \qquad s_k^{\text{OBD}} = \frac{1}{2}\mathbf{G}_{kk}(\boldsymbol{\theta})\boldsymbol{\theta}_k^2 \tag{5.6}$$

5.1.2.3 Optimal Brain Surgeon (OBS)

Around the same time as OBD, Hassibi & Stork (1992) also proposed a pruning method based on the quadratic model of the loss to solve the minimisation problem given in Equation 5.2 but uses the Lagrangian formulation to include the constraint to the solution of the minimization problem. Since OBS requires to compute the inverse of $\mathbf{H}(\boldsymbol{\theta})$, several approximations have been explored in the literature, including diagonal, as in the original OBS, Kronecker-factored (Martens & Grosse, 2015) as in ML-Prune (Zeng et al., 2021), or diagonal, but in an Kronecker-factored Eigenbasis (George et al., 2018), as in EigenDamage (Wang et al., 2019).

To pursue our analysis on pruning criteria, OBD is selected for the remaining in this chapter to represent second-order loss-modelling. Note that everything presented in the following sections can also be used in OBS-based methods.

5.1.2.4 Linear (LM) and Quadratic (QM) Models

For OBS and OBD to hold true, one major assumption is made: the training has converged. In addition to the Hessian being positive semi-definite, this assumption implies that first-order interactions are neglected because the training has converged thus the gradients should be zero. However, the current training strategies for deep

neural networks involve the use of regularisation techniques such as early stopping, Batch Normalisation (Ioffe & Szegedy, 2015) or Dropout (Srivastava et al., 2014) to counteract over-fitting. Therefore there is no reason to assume that the training has entirely converged, implying that the linear term should not be neglected. This phenomenon has been explored in the literature for OBS (Singh & Alistarh, 2020), as well as for structured pruning (Molchanov et al., 2019).

Because the gradient term – $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \Delta\boldsymbol{\theta}$ – no longer vanishes, it can be integrated to the quadratic model (QM), leading to the following saliency:

$$\Delta\mathcal{L}^{QM}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}_k) \approx \left| \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_k}^\top \Delta\boldsymbol{\theta}_k + \frac{1}{2}\mathbf{G}_{kk}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}_k^2 \right| \Rightarrow s_k^{\mathrm{QM}} = \left| -\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_k}\boldsymbol{\theta}_k + \frac{1}{2}\mathbf{G}_{kk}(\boldsymbol{\theta})\boldsymbol{\theta}_k^2 \right| \tag{5.7}$$

To avoid computing the Hessian approximation, one can use a simpler linear model (LM) to avoid the cost associated with computing second-order information to approximate $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$.

Only using first-order interactions, as introduced by Single-shot Network Pruning (Lee et al., 2019) – as demonstrated by Wang et al. (2020), lead to the following saliency:

$$\Delta\mathcal{L}^{LM}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = \left| \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \Delta\boldsymbol{\theta} \right| \quad \Rightarrow \quad s_k^{\mathrm{LM}} = \left| \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_k}\boldsymbol{\theta}_k \right| \tag{5.8}$$

5.1.3 Gradient based

5.1.3.1 Pruning Objective

Another important aspect of deep neural networks that one might want to preserve when pruning is signal propagation. Pruning can be applied at different stages of the training, at initialisation, early-on or mid training, or after training. No matter when the pruning is applied, it is always followed by a retraining phase. When parameters are removed from the neural network, a risk is to damage the gradient

propagation preventing the pruned network to retrain properly. Thus to ensure a good re-trainability of the pruned model, the pruning criterion should preserve parameters that guarantee a good flow of the gradient signal.

Gradient-based criteria are designed to preserve or improve the flow of the gradient between the original (dense) and pruned (sparse) network. These methods try to find a step $\Delta\boldsymbol{\theta}$ such that the flow of the gradient – $\|\mathbf{g}(\boldsymbol{\theta})\| = \left|\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}^\top \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|$ – in the pruned network is preserved or sometimes improved, leading to the following pruning objective:

$$\underset{\Delta\boldsymbol{\theta}}{\text{minimize}} \quad \Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\| \stackrel{\text{def}}{=} |\|\mathbf{g}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})\| - \|\mathbf{g}(\boldsymbol{\theta})\|| \tag{5.9}$$

The flow of the gradient signal can be estimated in regards to the training data (GraSP), or more generally, independently from any data (SynFlow).

5.1.3.2 Gradient signal preservation (GraSP)

Gradient-based pruning criteria were first introduced by Wang et al. (2020) in the context of pruning at initialisation. Previous methods to prune before training were solely based on loss-modelling criteria – SNIP (Lee et al., 2019). But the authors argued that at initialisation the loss is noisy and not very informative, and monitoring the signal propagation constitutes a more robust candidate to estimate the importance of parameters.

In their criterion, Wang et al. (2020) proposed to ensure good re-trainability by maximising the flow of the gradient in the pruned network by preserving parameters that lead to a greater reduction of the loss, thus accelerating the training. Larger gradient norm indicates that the contribution of the parameters to reduce the loss is higher:

$$\Delta\mathcal{L}(\boldsymbol{\theta}) = \lim_{\epsilon\to 0} \frac{L(\boldsymbol{\theta} + \epsilon\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}) - L(\boldsymbol{\theta})}{\epsilon} = \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}^\top \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} \tag{5.10}$$

Thus, parameters that impact negatively the gradient norm can be removed to achieve faster training, ie. greater loss reduction. To do so, the authors proposed the following criteria:

$$\Delta\mathcal{L}(\boldsymbol{\theta}+\delta) - \Delta\mathcal{L}(\boldsymbol{\theta}) = \frac{\partial\mathcal{L}(\boldsymbol{\theta}+\delta)}{\partial\boldsymbol{\theta}}^{\top}\frac{\partial\mathcal{L}(\boldsymbol{\theta}+\delta)}{\partial\boldsymbol{\theta}} - \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}^{\top}\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} \tag{5.11}$$

$$= \|\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\| + 2\delta\frac{\partial^2\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} - \|\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\| + \mathcal{O}(\|\delta\|_2^2) \tag{5.12}$$

$$\tag{5.13}$$

where δ is an infinitesimal perturbation to the weights. Thus we obtain:

$$s_k^{\text{GraSP}} = -\boldsymbol{\theta} \odot \mathbf{H}(\boldsymbol{\theta})\mathbf{g} \tag{5.14}$$

In this configuration, the Hessian $H(\boldsymbol{\theta})$ can be approximated using the Generalised Gauss-Newton $G(\boldsymbol{\theta})$. GraSP was initially designed to be applied at initialisation but the criterion can easily be applied anytime during training. However, the risk of applying GraSP later in training is that some parameters might have converged resulting in close to zero gradients.

5.1.3.3 Synaptic Flow (SynFlow)

Tanaka et al. (2020) identified a major bottleneck for pruning criteria. When pruning at initialisation, most criteria appears to be unstable for high pruning ratios and tend to damage the architecture beyond recovery, a phenomenon known as layer collapsing and previously discussed in Chapter 4. The authors identified that there exists a *Maximal Critical Compression* for which a network can be pruned without suffering from layer collapse, however most unstructured pruning criteria often fail at matching the maximal compression removing parameters that should not have been removed. To mitigate this issue, they proposed a data-agnostic pruning approach that preserves the synaptic flow by iteratively removing parameters that produce the least salient synaptic response:

$$s_k^{\text{SynFlow}} = \frac{\partial \mathcal{R}_{sf}}{\partial \boldsymbol{\theta}} \odot \boldsymbol{\theta} \quad \text{where} \quad \mathcal{R}_{sf} = \mathbb{1}^\top \left(\prod_{l=1}^{L} \left| \boldsymbol{\theta}^{|l|} \right| \right) \mathbb{1} \tag{5.15}$$

where L represent the layers of the network and $\mathbb{1}$ is an all-ones input vector. SynFlow can reach maximal critical compression and by preserving the global synaptic strength of a deep neural network, it can be seen as a generalised magnitude approach that preserves the individual synaptic strength.

5.2 Locality assumption

Whether *loss-modelling* or *gradient-based*, pruning criteria are *local* approximations of the current network state. Therefore, local loss or gradient approximations are generally only faithful in a small neighbourhood of the current parameters. Explicitly showing the terms that are neglected in pruning criteria, we have for loss modelling criteria:

$$\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = \Delta\mathcal{L}^{LM}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^2) = \Delta\mathcal{L}^{QM}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^3) \tag{5.16}$$

$$\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = \Delta\mathcal{L}^{OBD}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^3) \tag{5.17}$$

and for gradient-based criteria:

$$\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\| = \Delta\mathbf{g}^{GraSP}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^2) \tag{5.18}$$

$$\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\| = \Delta\mathbf{g}^{SynFlow}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^2) \tag{5.19}$$

Thus, when approximating $\Delta\mathcal{L}$ with $\Delta\mathcal{L}^{LM}$, or $\Delta\|\mathbf{g}\|$ with $\Delta\mathbf{g}^{GraSP}$ or $\Delta\mathbf{g}^{SynFlow}$, we neglect the terms in $\mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^2)$, and when approximating $\Delta\mathcal{L}$ with $\Delta\mathcal{L}^{QM}$ or $\Delta\mathcal{L}^{OBD}$, we neglect the terms in $\mathcal{O}(\|\Delta\boldsymbol{\theta}\|_2^3)$. As a consequence, criteria approximations are only valid in a small neighbourhood of $\boldsymbol{\theta}$, and are extremely likely to be

wrong when $\|\Delta\boldsymbol{\theta}\|_2$ is large.

To ensure that the saliency scores computed by the pruning criterion are correct, the locality of the function approximated needs to be preserved for the relevant parameters to be removed. Many factors can cause a large $\|\Delta\boldsymbol{\theta}\|_2$ – removing a large portion of the parameters at once, large magnitude of the parameters – but some tricks can be applied to prevent $\|\Delta\boldsymbol{\theta}\|_2$ from being too massive.

5.2.1 Multi-stage pruning

When pruning, $\|\Delta\boldsymbol{\theta}\|_2$ can be large when a large portion of the parameters is pruned at once. In practice to reduce the memory and computational footprint of a model and obtain an efficient pruned network, one needs to apply high ratios of pruning, usually around 90% for it to be memory efficient (see Chapter 4, Section 4.1.5). If we were to remove 90% parameters at once – one-shot pruning – the function approximated by the pruning criterion would likely be wrong.

An easy fix to mitigate this issue discussed in Chapter 4 is to adopt a *multi-stage* pruning approach. Under multi-stage pruning framework, only a small number of parameters is pruned at a time, re-estimating the underlying model between each stage. A parameter with a low importance score might become important once a specific set of parameters are removed. By removing parameters in small batches and re-estimating the model between each pruning phase, we are able to capture more granularity.

This is similar to what is done in the lottery ticket hypothesis (Frankle & Carbin, 2018) pruning framework, except that we do not retrain the model in between each pruning stage, we only re-estimate the underlying model of the criterion. However, without fine-tuning phases in between the different pruning stages, this strategy violates the *convergence* assumption behind OBD and OBS, since after the first stage of pruning the network is no more at convergence. It is also worth noting that criteria that do not rely on an underlying model such as MP, pruning in *one-shot* or *multi-stages* are equivalent.

The number of stages, which is denoted by π, is typically overlooked in the literature despite being an important factor when pruning. Both Zeng et al. (2021) and Wang et al. (2019) use only 6 stages of pruning, Tanaka et al. (2020) uses 100 stages. Multi-stage pruning is a great way to prevent layers collapsing in pruning.

The sparsity at each pruning phase can be increased following either a *linear schedule*, where each step prunes the same number of parameters, or an *exponential schedule*, where the number of parameters pruned at each stage gets smaller and smaller (see Figure 5.1). The fewer parameters remaining, the more we should be careful about which parameters are removed. For that reason, exponential scheduling is typically used in the literature (Zeng et al., 2021; Wang et al., 2019; Frankle & Carbin, 2018; Renda et al., 2020).

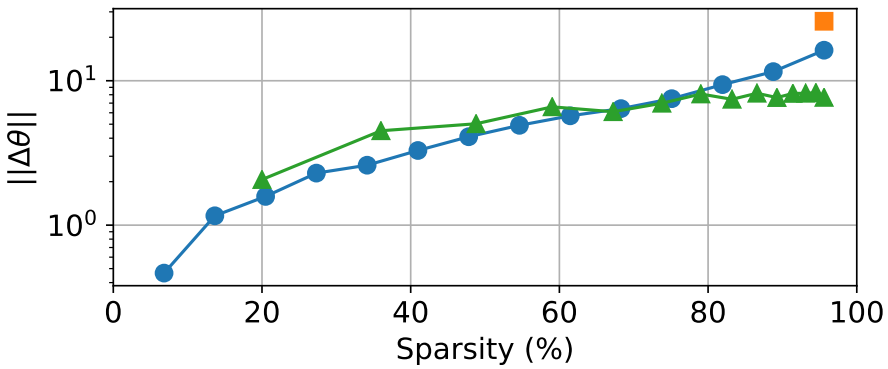


Figure 5.1: $\|\Delta\boldsymbol{\theta}\|_2$ for different level of sparsity and multi-stage schedule strategies, oneshot (orange square), linear (blue dots), exponential (green triangles).

5.2.2 Constraining step-size

In the optimisation literature, more specifically in trust-region methods (Nocedal & Wright, 2006), it is common to add a penalty component to a problem to constrain it into a feasible region where we know the problem has a solution. This can be extended in the context of pruning to enforce locality in quadratic models. We can enforce locality by penalising the pruning criterion when it decides to take steps that are too large by adding a norm penalty $\|\boldsymbol{\theta}\|_2^2$ constraint, in order to stay in a region where we can trust the model. This can be done by adding a norm penalty, $\frac{\lambda}{2}\|\boldsymbol{\theta}_k\|_2^2$, to the saliencies computed by any criterion:

$$s = s_{\mathcal{P}}(\boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}_k\|_2^2 \tag{5.20}$$

where $s_{\mathcal{P}}$ is the criterion saliency and λ is a hyper-parameter to control the strength of the constraint. Note that a small value of λ leaves the saliencies unchanged, while a large value of λ transforms any pruning criterion into Magnitude Pruning (Equation 5.1).

5.2.3 Others considerations

Finally, $\|\Delta\boldsymbol{\theta}\|_2$ can be large if $\boldsymbol{\theta}$ is large itself. This is dependent on the training procedure of the network but can be easily mitigated by constraining the norm of the weights through regularisation tools. This can be done using either L_2 regularisation, weight decay, or both. Since nowadays weight decay is almost systematically used by default when training networks (e.g. He et al. (2016b); Xie et al. (2017); Devlin et al. (2018)), this will not be studied further.

In summary, to ensure that the locality assumption behind the underlying model used by the pruning criterion is respected, a combination of pruning stages π and regularisation λ can be applied to the saliency score to better solve the pruning objective – preserving the loss or gradient flow.

5.3 Pruning Integrity

Despite the wide variety of pruning approaches, a universal method that consistently achieves state-of-the-art performance has yet to be identified (Frankle et al., 2021). While currently proposed pruning criteria perform better than random, there is no single criterion that outperforms all the others, rising interrogations behind the integrity of pruning criteria design and evaluation. Across a wide range of pruning ratios, all criteria obtain accuracy performances that lie in a close region after fine-tuning, and one needs to reach extreme pruning ratios to start seeing significant

differences between pruning criteria (see Figure 5.2). To research how to improve upon existing heuristics, we first need to assess the integrity of existing criteria and understand which part of the current pruning criteria design is accountable for good performance results.

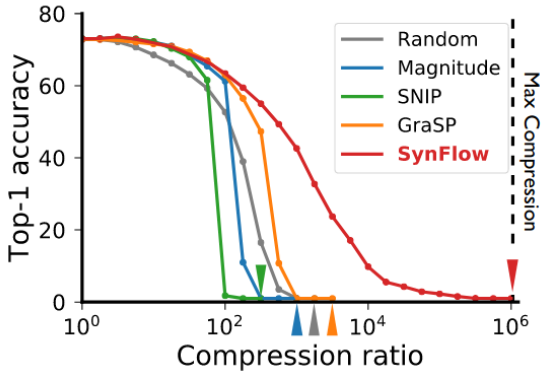


Figure 5.2: Pruning accuracy for different pruning methods applied at initialisation. We can observe that all pruning criteria lie in close range. For information a 10^2 corresponds to 99% pruning ratio. Reproduced from Tanaka et al. (2020).

Recent efforts in the pruning literature have started to investigate the relationship between different pruning criteria families (Lubana & Dick, 2021). However, the link between the pruning objective – ie. what properties of the network the criterion is trying to preserve – and the performance of the pruned model is often overlooked. In practice pruning is always followed by a fine-tuning step, thus most papers focus on the best accuracy after fine-tuning and the highest compression ratio they are capable to reach to evaluate their criterion performance. Ultimately these two factors are the ones we care about, but most papers presenting new pruning criteria omit to assess the quality of a pruning criterion in itself. Fine-tuning plays an important role in pruning, but it brings all the stochasticity around *training* deep neural networks – hyper-parameters selection, choice of initialisation, training scheme etc. – that is yet to be fully understood. Therefore both, the foundation of the pruning criterion and fine-tuning, should be considered and evaluated independently.

There are three major families of pruning criteria: loss-preservation, gradient-preservation and magnitude. If pruning criteria were to be efficient, it would mean that pruned models better at minimising the change in loss, or gradient flow, obtain

better performances after fine-tuning the pruned network – better re-training performance. To assess the rightfulness of each criterion design, we reassure locality of the underlying model needs. For that, pruning is performed in multiple stages π, and a norm penalty λ is added to the saliency for each pruning criterion to be as good as possible at minimising the loss or gradient flow.

For better clarity in assessing pruning criteria integrity, each criterion will be referred by its acronym: MP for Magnitude Pruning (equation 5.1), LM for Linear Model (equation 5.8), QM for Quadratic Model(equation 5.7), OBD for Optimal Brain Damage (equation 5.6), GraSP for Gradient Signal Preservation (equation 5.14) and finally, SynFlow for Synaptic Flow (equation 5.15).

5.3.1 Performance Metrics

5.3.1.1 Pruning Objective

The main goal of pruning is to produce a sparse network with an optimal capacity by shrinking the number of parameters in a dense neural network while maintaining, and ideally improve, overall prediction and generalisation performances. Thus, practitioners are mainly interested in the trade-off between *ratio of compression* and *accuracy performance*.

A pruning criterion estimates the importance of parameters based on their influence for solving a particular pruning objective (preserving the original network loss, or gradient flow). To assess the integrity of a criterion, two factors need to be checked: (1) whether the criterion is solving the pruning objective, and (2) whether optimising for that particular pruning objective lead to better performance after fine-tuning. Thus, pruning criteria will be evaluated regarding their ability to preserve the original network function before/after pruning (equation 5.2, 5.9), and their accuracy performance after fine-tuning the sparse network.

The preservation of the *loss function* is measured as follows:

$$\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = |\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta})| \tag{5.21}$$

while,

$$\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\| = \|\mathbf{g}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})\| - \|\mathbf{g}(\boldsymbol{\theta})\| \tag{5.22}$$

is used to quantify the change in *gradient flow* between the original and pruned model. To measure the change in performance, the validation error gap both before and after fine-tuning is used.

$$\Delta\mathbf{error} = \mathbf{error}_{sparse} - \mathbf{error}_{dense} \tag{5.23}$$

where,

$$error = (1 - accuracy) = 1 - \frac{corrects}{total} = 1 - \frac{TP + TN}{TP + TN + FP + FN} \tag{5.24}$$

with TP being true positive, TN true negative, FP false positive, FN false negative. Note that the top-1 accuracy score is measured in our experiments.

5.3.1.2 Correlation with end accuracy

Pruning criteria are designed to remove parameters based on their level of importance derived from some pruning objective. For a pruning criterion to be efficiently designed, better solving the pruning objective – in the case of unstructured pruning, preserving the loss or gradient flow – should lead to better performances after the fine-tuning phase. To assess the relationship between pruning objective (eq. 5.21,eq. 5.22) and performance accuracy (eq. 5.23), the correlation between both part is computed using Spearman rank correlation:

$$\rho_{spearman} = 1 - \frac{6\sum_i d_i^2}{n(n^2 - 1)} \tag{5.25}$$

A non-parametric correlation measure is used to assess the strength of the relationship between the different pruning objectives and the end performance as it

is the ordinal order that is of interest and not the numerical value in themselves. A correlation close to $+1$ or -1 would suggest that there is a positive, or negative, association of ranks, ie. best preserving the original network function transfer to better accuracy after fine-tuning. On the other hand, a 0 correlation score will indicate that no meaningful relationship between the two variables can be observed.

5.3.2 Methodology

Pruning can be applied at multiple stages during the training. In Frankle et al. (2021), the authors provided an extensive investigation and comparison of the performance accuracy of different pruning methods when pruning is applied at initialisation and end of training. They observed that, when pruning at initialisation, the sparsity per layer seemed to be what matters the most, and not the pruning masks, suggesting that all pruning criteria were equal. For that reason, to assess the integrity of the different pruning criteria, pruning after training is considered in this study.

Algorithm 1 presents the pruning framework used in this work. The neural network is first trained, then pruning is applied. Multiple stages of pruning $\pi \in [1, 14, 140]$ are performed following an exponential schedule (see Figure 5.1), with different norm regularisation factors $\lambda \in [1e^{-4} \dots 1e^{100}]$ applied to the saliency, the idea is to obtain a range of pruned model with different values of loss, or gradient flow, minimisation to compare whether better preserving the original model function lead to better performance. Finally, fine-tuning is performed in a single phase using the same hyper-parameters as for the original training with learning rate rewinding. *Global* pruning is used for all the criteria. Note that while the fine-tuning phase would require hyper-parameter optimisation, Renda et al. (2020) showed that using the same ones as for the original training usually leads to good results. The hyper-parameters used in this set of experiments are provided in Appendix B.1.

The experiments to assess the integrity of pruning criteria are conducted on three different network architectures to test a variety of deep learning models with

Algorithm 1 Pruning Framework

Require: Network $f_{\boldsymbol{\theta}}$ with $\boldsymbol{\theta} \in \mathbb{R}^D$, dataset $\mathcal{D}$, number of pruning iterations π, norm regularisation λ and sparsity κ.

$f_{\boldsymbol{\theta}} \leftarrow \text{Training}(f_{\boldsymbol{\theta}}, \mathcal{D})$
$\kappa_0 \leftarrow 0$
$\mathbf{m} \leftarrow \mathbf{1}^D$
for $i = 1$ to π **do**
    $\kappa_i \leftarrow \kappa_{i-1} + \frac{(\kappa - \kappa_0)}{\pi}$ **or** $\kappa_i \leftarrow \kappa_{i-1} + (\kappa - \kappa_0)^{i/\pi}$ ▷ Compute sparsity for iteration i
    $\boldsymbol{s} \leftarrow \text{Saliencies}(f_{\boldsymbol{\theta} \odot \mathbf{m}}, \lambda, \mathcal{D})$ ▷ Compute saliencies (Equation 5.1, 5.6, 5.7, 5.8, 5.14 or 5.15).
    $\mathbf{m}[\text{argsort}(\boldsymbol{s})[: \kappa_i D]] \leftarrow 0$ ▷ Mask the parameters with smallest saliencies.
$f_{\boldsymbol{\theta} \odot \mathbf{m}} \leftarrow \text{Training}(f_{\boldsymbol{\theta} \odot \mathbf{m}}, \mathcal{D})$ ▷ Optional fine-tuning
return $f_{\boldsymbol{\theta} \odot \mathbf{m}}, \mathbf{m}$

good performance for different parameters capacity: an MLP on MNIST, and with both VGG11 (Simonyan & Zisserman, 2015) without batch normalisation and a pre-activation residual network 18 (He et al., 2016b) on CIFAR10 (Krizhevsky et al., 2012). All models architectures are described in chapter 2. As a quick reminder, an MLP is a fairly small model with just a few hundred thousand parameters, that is easy to supervise containing only fully connected layers. On the other hand, VGG architectures introduce convolutional layers and constitute models that are highly over-parametrised known to be easily prunable. Finally, pre-activation residual networks are computationally more complex than VGG with their skip connections, making unstructured pruning more challenging. More details on the splits, data augmentation and hyper-parameters can also be found in Appendix B.1. Note that three to five different random seeds are used, and both mean and standard deviations are reported. ImageNet and Resnet50 architectures are also used to assess the scalability of the different claims by validating that observations on CIFAR10 hold on larger network architectures and datasets. Note that due to compute limitations, this set of experiments was only performed with loss-modelling criteria.

The integrity of different pruning criteria is assessed in two times:

1. Before fine-tuning – just after pruning, *how well does a criterion preserve the original network function – loss or gradient – in the pruned network* (pruning

objective)

2. After fine-tuning the pruned network, whether *better preserving the original (dense) network property lead to better performances after fine-tuning* (pruning performance)

Both will be investigated for each pruning family: loss-preservation and gradient-preservation. Magnitude heuristic will be used as a baseline comparison. Details on implementations for the different criteria can be found in appendix B.1.

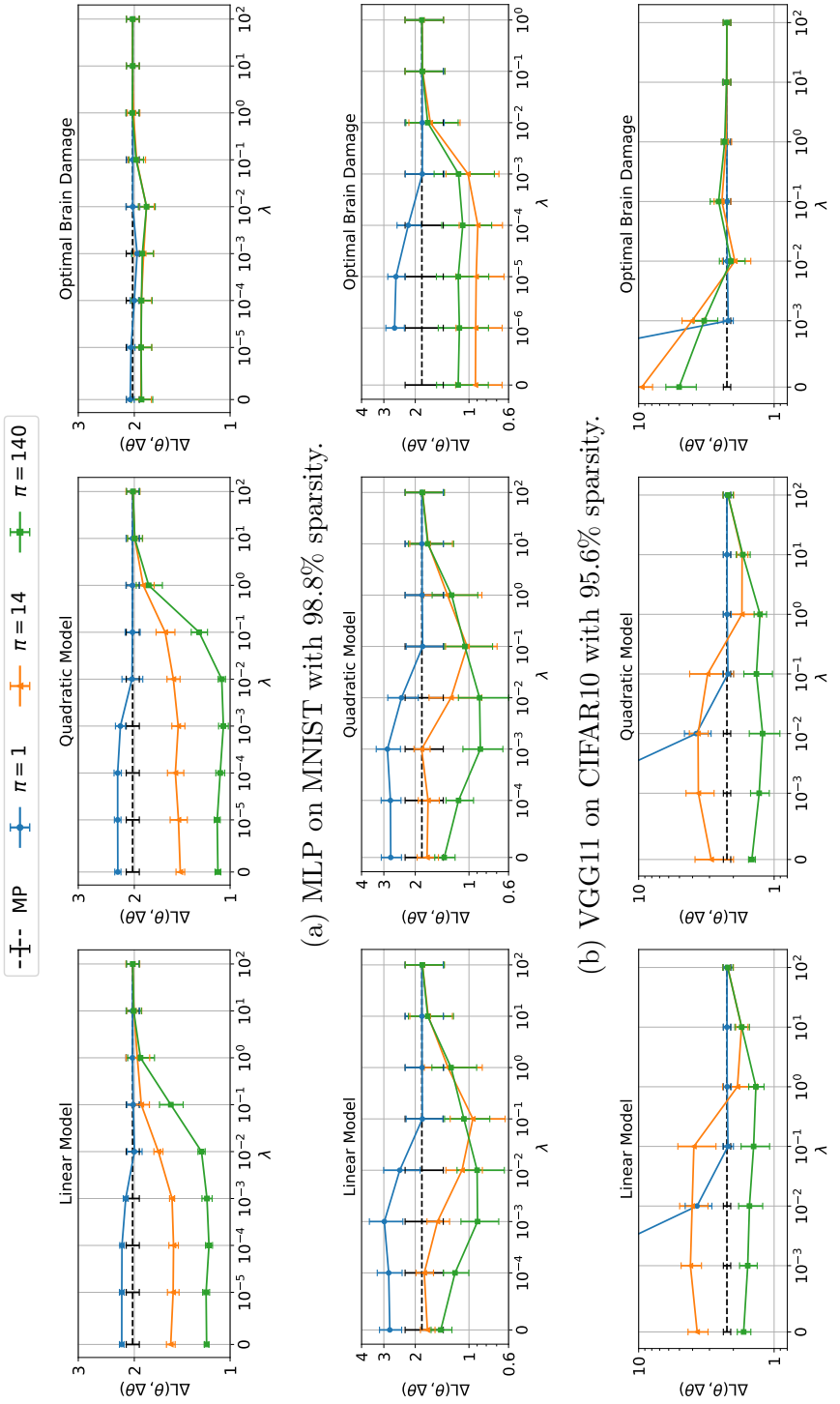
5.4 Experimental results

5.4.1 Loss-preservation criteria

For loss-preservation criteria, the pruning objective is to minimise the change in loss (equation 5.21) between the original (dense) network and the pruned (sparse) network. The criteria considered for this section are: LM (equation 5.8), QM (equation 5.7) and OBD (equation 5.6), in addition to MP (equation 5.1) as baseline.

5.4.1.1 Before Fine-tuning

The first step to investigate how good are pruning criteria is to measure their capacity at preserving the original network function. The quality of different loss-modelling pruning methods is evaluated by measuring their preservation capabilities $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. To obtain the best possible results, the impact of enforcing locality is studied as a function of λ, for different numbers of pruning stages π, where λ is our step-size norm regularisation factor. Note that typical usage of these criteria in a one-shot setting would be with a regularisation strength $\lambda = 0$ and a number of pruning stages $\pi \approx 1$. MP, the baseline which is invariant to both λ and π, is also reported. For reference, the original dense networks reached a validation error rate before pruning of 1.47 ± 0.04 % for the MLP, 10.16 ± 0.29 % for VGG11 and 4.87 ± 0.04 % for the PreActResNet18.



(a) MLP on MNIST with 98.8% sparsity.

(b) VGG11 on CIFAR10 with 95.6% sparsity.

(c) PreActResNet18 on CIFAR10 with 95.6% sparsity.

Figure 5.3: $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ for different number of pruning stages π, as a function of λ, the step size constraint strength, using either (left) LM, (middle) QM or (right) OBD criteria. MP, which is invariant to λ and to the number of pruning stages, is displayed in dashed black. The curves are the mean and the error bars the standard deviation over 5 random seeds. OBD with $\pi = 1$ and $\lambda = 0$ diverged for all of the 5 seeds. Increasing the number of pruning stages drastically reduces $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. A $\lambda > 0$ can also help improving performances.

Enforcing locality Figure 5.3 presents the loss preservation capabilities for different pruning criteria (column) as a function of different level of locality enforcement π and λ. It shows that increasing the number of pruning stages π can drastically reduce $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ when using LM, QM and OBD criteria. It demonstrates the importance of applying local steps when pruning. Constraining the step size through $\frac{\lambda}{2}\|\boldsymbol{\theta}_k\|_2^2$ (λ) can also further reduce $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$, on CIFAR10 in particular. The trend, however, is less pronounced on MNIST. A possible explanation is that it is due to the pruning step size: the MLP contains 260k parameters verses 9.7M for VGG11, so the number of parameters pruned at each stage in VGG11 is still large, even with $\pi = 140$. This translates to a bigger $\|\Delta\boldsymbol{\theta}\|_2$ that needs to be controlled by the regularisation constraint.

When performing the pruning in several stages, LM and QM reach better loss minimisation performances than OBD criterion. Without retraining phases between pruning stages, the convergence assumption behind OBD is violated. This is however not the case for LM and QM. Note that OBD still works reasonably well on VGG11. This could be related to the large number of parameters of the model.

Preservation capabilities Table 5.1 contains the best $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ obtained for each of the network architectures and loss-modelling pruning criteria. The best performance observed amongst all the λ and π tested are reported. Criteria that model the loss, LM and QM in particular, are better at preserving the loss before/after pruning than MP. For comparison, gradient-based criteria scores at preserving the loss are also reported in Table 5.1.

Network	$\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$					
	MP	**OBD**	**LM**	**QM**	**GraSP**	**SynFlow**
MLP	2.0 ± 0.1	1.8 ± 0.1	1.2 ± 0.1	**1.1 ± 0.1**	2.1 ± 1.4	1.8 ± 0.1
VGG11	1.8 ± 0.4	**0.9 ± 0.2**	**0.9 ± 0.2**	**0.9 ± 0.2**	14.8 ± 9.5	1.0 ± 0.1
PreActResNet18	2.2 ± 0.1	2.0 ± 0.5	1.4 ± 0.2	**1.2 ± 0.3**	6941 ± 9814	2.7 ± 0.1

Table 5.1: Summary of the best $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ across values of λ for different networks and pruning criteria, with $\pi = 140$. QM achieves better loss-preservation than other criteria. OBD performs worse than QM, since we violate its convergence assumption when pruning in several stages.

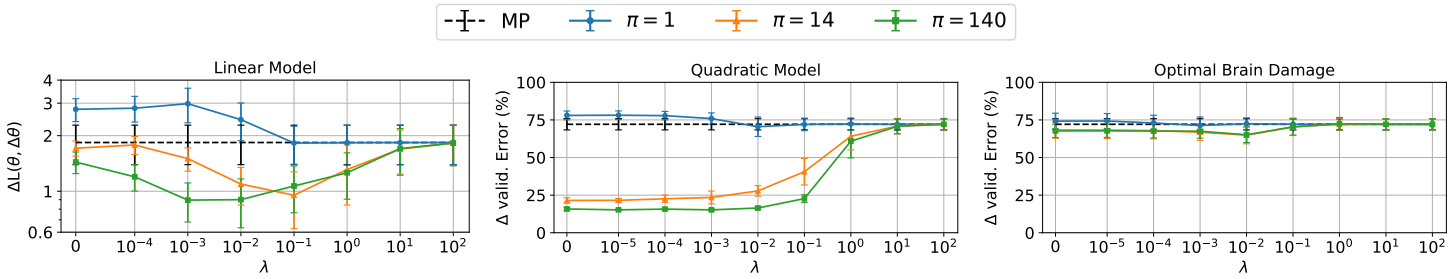
Network	Gap of Validation Error (%)			
	MP	**OBD**	**LM**	**QM**
MLP	72.09 ± 3.72	64.89 ± 5.74	**16.35 ± 0.77**	**15.22 ± 0.62**
VGG11	56.19 ± 17.9	18.84 ± 5.54	**5.89 ± 1.52**	**5.92 ± 2.14**
PreActResNet18	74.13 ± 4.59	49.08 ± 8.18	26.79 ± 8.61	**21.48 ± 5.96**

Table 5.2: Best validation error gap **before/after pruning** for different networks and pruning criteria.

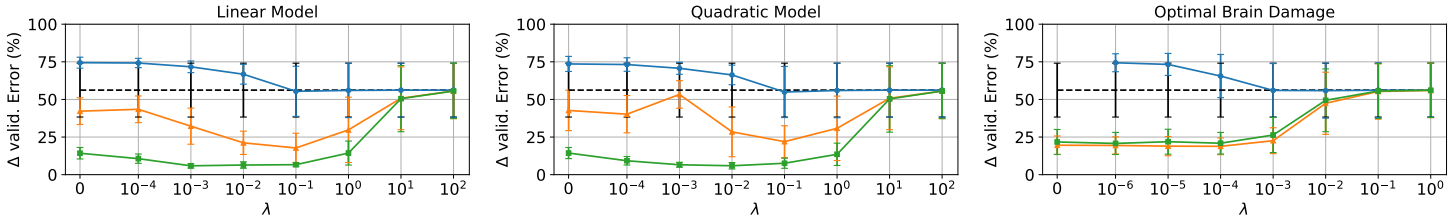
Validation error gap before/after pruning A first step to check the integrity of loss-modelling pruning criterion is to verify if better preserving the loss between the dense and sparse model lead to smaller validation error gaps right *after pruning*, before any fine-tuning of the pruned network. Mirroring Figure 5.3, Figure 5.4 presents the validation error gap before/after pruning as a function of different level of locality enforcement π (number of pruning stages) and λ (norm regularisation). Enforcing locality and thus better preserving the loss between the original and pruned model, lead to better performances – smaller validation error gap – right after pruning. Those results partly confirm the intuition behind loss-modelling pruning objective that better preserves the loss function, create better-performing pruned models before/after pruning. Note that the pruned models still suffer from a significant loss in accuracy compared to the dense network. A retraining (fine-tuning) phase is required to best recover the original model accuracy. Best gap in validation error before/after pruning are summarised in Tables 5.2.

5.4.1.2 After Fine-tuning

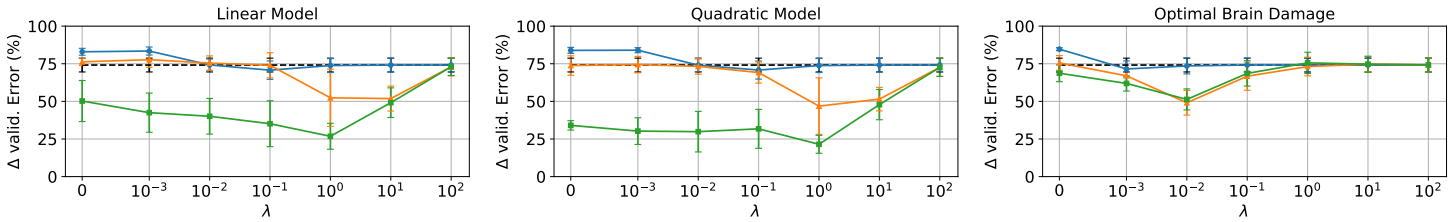
Having checked that Loss-modelling criteria do well at preserving the loss function before/after pruning, and that minimising the change in loss lead to smaller validation error gaps after pruning, the question remaining is whether previously observed results hold after fine-tuning the pruned model. In other words, if better preserving the loss lead to better accuracy performance after fine-tuning. For that, the correlation between the pruning objectives $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ and the gap in validation error is monitored. Fine-tuning is a mandatory step in pruning deep neural networks, for



(a) MLP on MNIST with 98.8% sparsity.



(b) VGG11 on CIFAR10 with 95.6% sparsity.



(c) PreActResNet18 on CIFAR10 with 95.6% sparsity.

Figure 5.4: Same as Figure 5.3, but displaying the validation error gap before fine-tuning. With proper number of pruning stages and step size regularization, LM and QM can produce pruned networks that are drastically better than the ones pruned using MP.

loss-modelling pruning to be efficient, a criterion that best minimises the change in loss should, in theory, obtain better performances.

Correlation between preservation and performance To assess loss-modelling criteria integrity, all the different models obtained with various levels of locality enforcement are considered to provide a range of models with different loss-preservation scores. Figure 5.5 displays scatter plots of all the experiments ran, to observe how well loss-preservation (x-axis) correlates with performance after fine-tuning (y-axis). Surprisingly, although it is possible to obtain networks with smaller $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$, and thus better-performing networks right after pruning, the performances after fine-tuning do not correlate significantly with the gap in validation error. Except for the MLP on MNIST, whose Spearman's rank correlation coefficient is $\rho = 0.67$, there is only a weak correlation between $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ and the validation error gap after fine-tuning ($\rho = 0.27$ for VGG11 and $\rho = 0.20$ for PreActResNet18).

Hyper-parameters selection Training can be very sensitive to hyper-parameters. To verify that these observations are not due to a specific choice of fine-tuning hyper-parameters, hyper-parameter grid search is performed. Three different learning rate (0.1, 0.01, 0.03) and three different l2-regularisation (0, 5e-4, 5e-5) are tested on LM, QM and MP on 5 different random seeds. In this set of experiments, $\lambda \in \{0, 0.01, 0.1, 1\}$ and $\pi \in \{14, 140\}$ are used. Figure 5.6 shows that optimising hyper-parameters for fine-tuning can lead to better performance after fine-tuning, but does not increases the correlation between the performances after fine-tuning and $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. The lack of correlation can thus not be explained by bad fine-tuning hyper-parameters. Models unable to re-train are not shown in this figure.

5.4.2 Gradient-flow preservation

No clear correlation can be observed from better preserving the loss function on the performance of the pruned model after fine-tuning for large deep neural networks. It seems that loss-modelling pruning criteria are not efficient. Another way to reduce the size of a model is to discard parameters that impact negatively the gradient

flow of the network. In this section, a similar exploration study is conducted on gradient-based criteria to evaluate whether preserving the gradient flow constitutes a better pruning objective, for this GraSP (eq. 5.14) and SynFlow (eq. 5.15) criteria are considered and compared to MP.

5.4.2.1 Before Fine-tuning

Gradient-based pruning objectives is looking at preserving the signal propagation between the original and pruned model. This can be evaluate by monitoring the change over the gradient flow $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ (eq. 5.22). As for loss-modelling criteria, the impact of enforcing locality (π and λ) is studied to obtain the best possible pruned network that minimises the change in $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$.

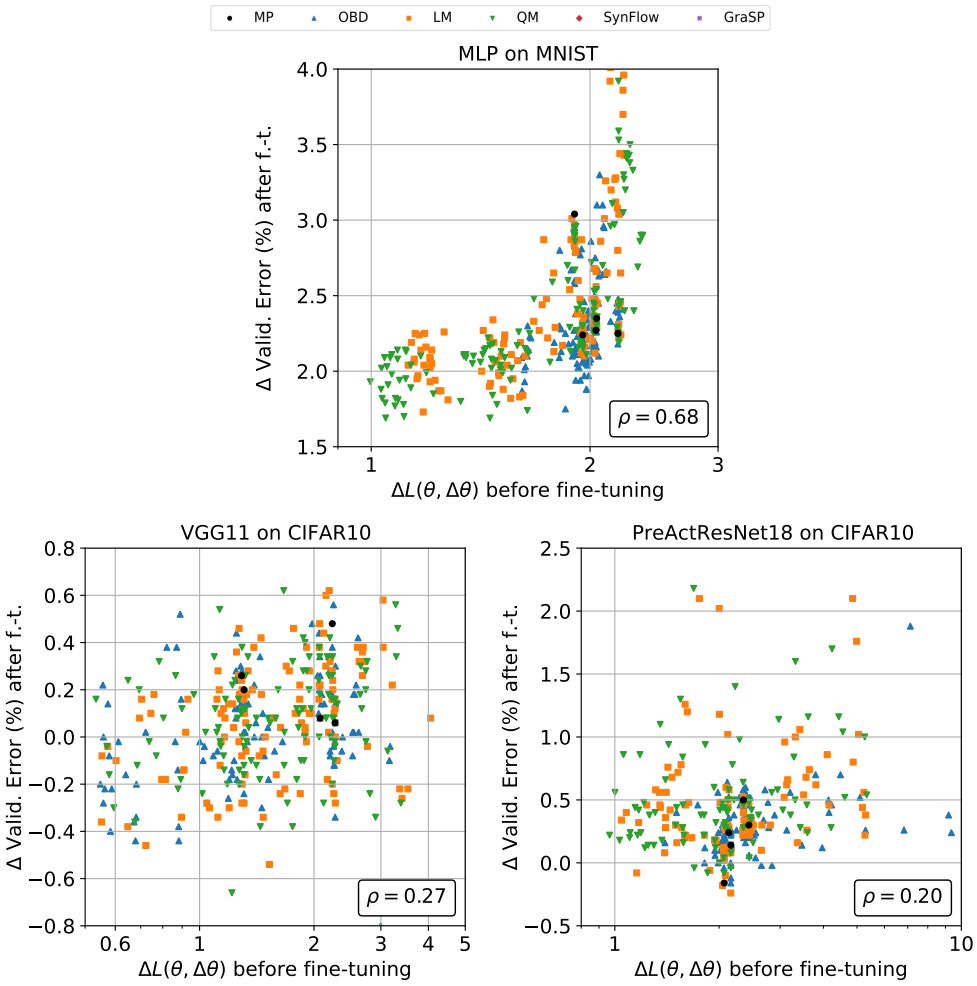


Figure 5.5: Gap of validation error after fine-tuning as a function of $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. We can observe a slight correlation on the MLP network architectures, however this does not translate to larger scale architectures VGG11 and PreActResnet18.

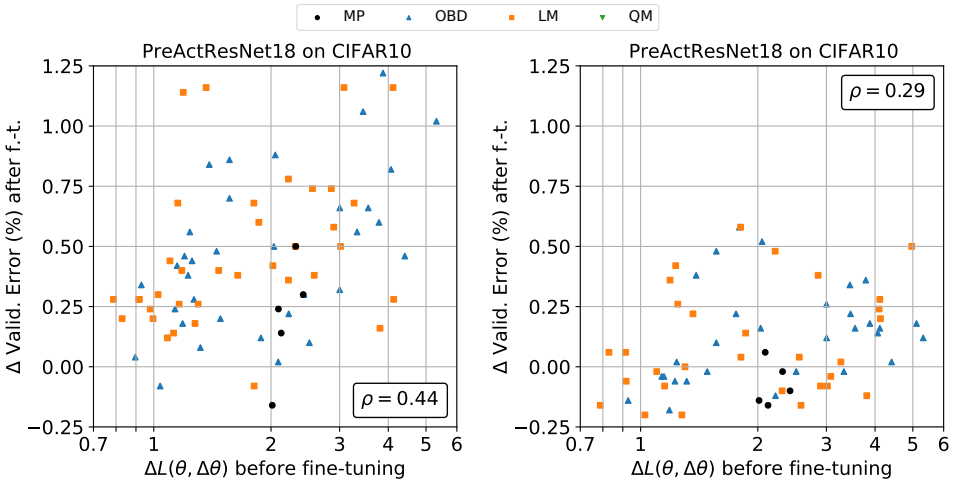
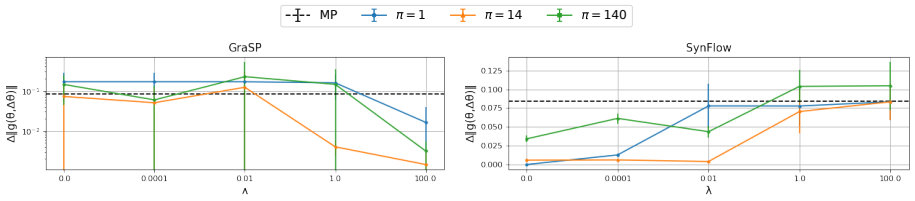


Figure 5.6: Left: Using the same hyper-parameters for fine-tuning as the ones of the original training. Right: Performing hyper-parameters optimisation for the fine-tuning.

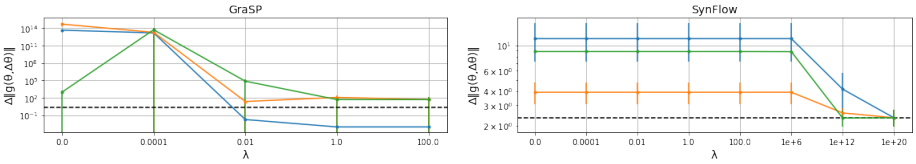
Enforcing locality Figure 5.7 pictures the impact of enforcing locality on gradient-based pruning metrics. The benefits of adding a norm penalty λ to the criterion, or performing multiple stages of pruning π are less pronounced on gradient-based criteria than on loss-modelling criteria (Figure 5.3). On larger network architectures (VGG11 and PreActResnet18), it appears that small λ have little effect on SynFlow criterion, and multiple stages of pruning do not contribute to improving GraSP preservation capabilities.

SynFlow observations are likely due to the fact that the heuristic estimates the gradient flow in a data-agnostic manner, which might lead to estimated gradients that have a norm drastically different from the approximate model, making it challenging to tune λ accordingly. Recall that SynFlow estimates the importance of each parameter based on their synaptic response by computing the gradient response from an all ones input entry vector (see eq. 5.15). Overall, performing the pruning in multiple steps π helps outperform one-shot pruning ($\pi = 1$), re-enforcing the importance of removing a small number of parameters at a time to better respect the locality assumption behind these pruning criteria.

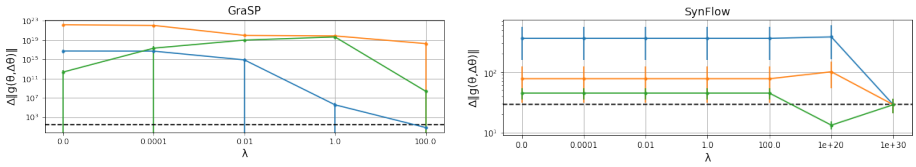
However, it is not the case for GraSP criterion where one-shot pruning outperforms multi-stage pruning. For GraSP, we observe that the λ regularisation is highly beneficial to help preserving $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. A possible explanation is in the



(a) MLP on MNIST with 98.8% sparsity.



(b) VGG11 on CIFAR10 with 95.6% sparsity.



(c) PreActResNet18 on CIFAR10 with 95.6% sparsity.

Figure 5.7: $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ for different number of pruning stages π, as a function of λ. As for the loss-modelling criteria, augmenting the number of π help to better preserve the gradient flow before/after pruning. λ regularisation however requires a more careful tuning as the magnitude of the saliencies issued from the gradient-based criteria can be high.

way GraSP criterion is designed. GraSP is arranged to remove parameters that impact negatively the gradient flow, hence trying to keep parameters that will improve $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. In this regard, enforcing locality helps better achieve the criterion original purpose, but augmenting the gradient flow is at risk of producing gradients that are too large, making it harder to re-train the model from an optimisation perspective. Adding a norm penalty helps mitigate this issue and keeps the gradient flow closer to the original network. Section 5.4.4 presented later in this chapter, explore a variant of GraSP criterion that aims at preserving the gradient flow instead of maximising it.

Preservation capabilities Similar to Table 5.1, Table 5.3 contains the best $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ obtained right after pruning for each pruning criteria. GraSP, SynFlow, and especially MP, are good at preserving the gradient flow before/after pruning. This may partly explain why despite its simplicity, MP obtains really good performance in practice compared to more complex pruning metrics.

Network	$\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$		
	MP	**GraSP**	**SynFlow**
MLP	3e-3 ± 4e-4	**1e-3 ± 1e-3**	**1e-5 ± 1e-6**
VGG11	2.4 ± 0.4	**1e-4 ± 1e-2**	8.8 ± 0.2
PreActResNet18	29 ± 7.9	**6.6 ± 1.0**	13 ± 2.1

Table 5.3: Summary of the best $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ across values of λ for different networks and pruning criteria. The best performing criteria varies a lot depending on the network architecture. GraSP and SynFlow are good at preserving the norm of the gradients.

Network	Gap of Validation Error (%)		
	MP	**GraSP**	**SynFlow**
MLP	72.09 ± 3.72	**60.41 ± 39.78**	**62.29 ± 4.81**
VGG11	56.19 ± 17.9	67.21 ± 18.5	**25.77 ± 6.65**
PreActResNet18	74.13 ± 4.59	**52.43 ± 28.43**	84 ± 1.03

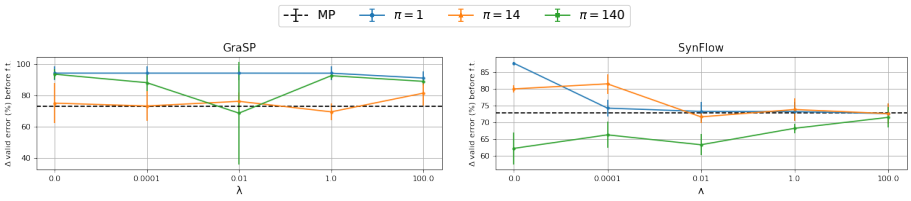
Table 5.4: Best validation error gap **before/after pruning** for different networks and loss-modelling pruning criteria.

Validation error gap before/after pruning Similar to Table 5.2, Table 5.4 contains a summary of the best gap in validation error obtain for both gradient-based criteria. Gradient-based pruning metrics are less efficient at minimising the gap in validation error before/after pruning than loss-modelling criteria (see Table 5.2). Moreover, best preserving the gradient flow (Table 5.3) doesn't necessarily lead to smaller gap in validation error. Note that the observations are also more noisy for GraSP in particular.

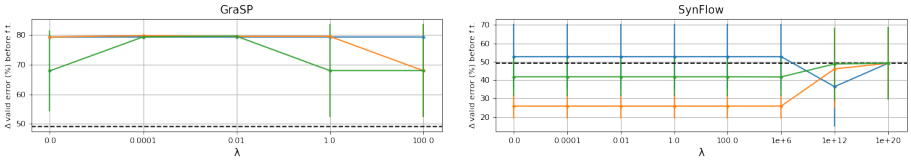
Figure 5.8 presents the validation error gap as a function of π and λ. Again, gradient-based pruning criteria are very noisy but with a good tuning of π and λ we are able to obtain good performing networks right after pruning, but still inferior to the one obtained with loss-modelling pruning criteria (see table5.2).

5.4.2.2 After Fine-tuning

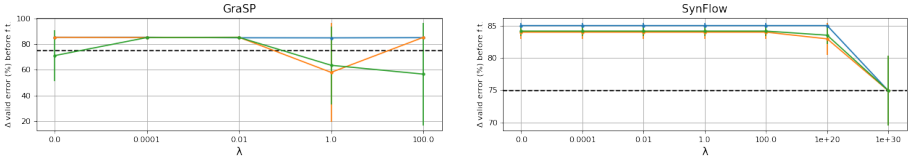
If the benefits from preserving the gradient flow between the original and pruned model on the performance right after pruning are less pronounced compare to preserving the loss function, preserving the norm of the gradients helps to preserve the network dynamic evolution (Lubana & Dick, 2021). Thus models that better pre-



(a) MLP on MNIST with 98.8% sparsity.



(b) VGG11 on CIFAR10 with 95.6% sparsity.



(c) PreActResNet18 on CIFAR10 with 95.6% sparsity.

Figure 5.8: Similar to Figure 5.7, but displaying the validation error gap before fine-tuning instead of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$.

serve the gradient flow have the potential to be better candidates to reach greater performance after fine-tuning.

Correlation between gradient preservation and performance To assess whether preserving the gradient-flow lead to better performing pruned networks after fine-tuning, the correlation between the pruning objective $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ and the end-accuracy is studied. Figure 5.9 displays scatter plots of all the experiments run using gradient-based criteria. As for loss-modelling criteria, multiple levels of locality enforcement are reported to provide a wide range of gradient-flow with diverse preservation scores. For the small MLP network architectures (Figure 5.9, left), a negative correlation score can be observed meaning that the better the gradient-flow is preserved, the worse the accuracy after fine-tuning ($\rho = -0.7$). Compare to larger networks, MLP models are trained to convergence, thus when pruning at the end of training most parameters have a gradient close to zero. If we try to preserve the gradient norm and remove parameters associated with zero-gradients, we might remove important connections. Targeting parameters that slightly increase the gradient flow, ensure good re-trainability of the pruned network.

For larger networks ((Figure 5.9, centre – VGG11, and right – PreActResnet18) a lot of experiments have a huge Δ validation error, suggesting that the sparse architecture was damaged beyond recovery during the pruning phase. In the case of VGG11, even when ignoring networks that failed to re-train, the correlation does not improve – from $\rho = 0.08$ to $\rho = 0.2$ (see Figure 5.10). There is a weak correlation on the PreActResnet18 ($\rho = 0.46$) resulting from very large gradient-flow being informative of poor network re-trainability.

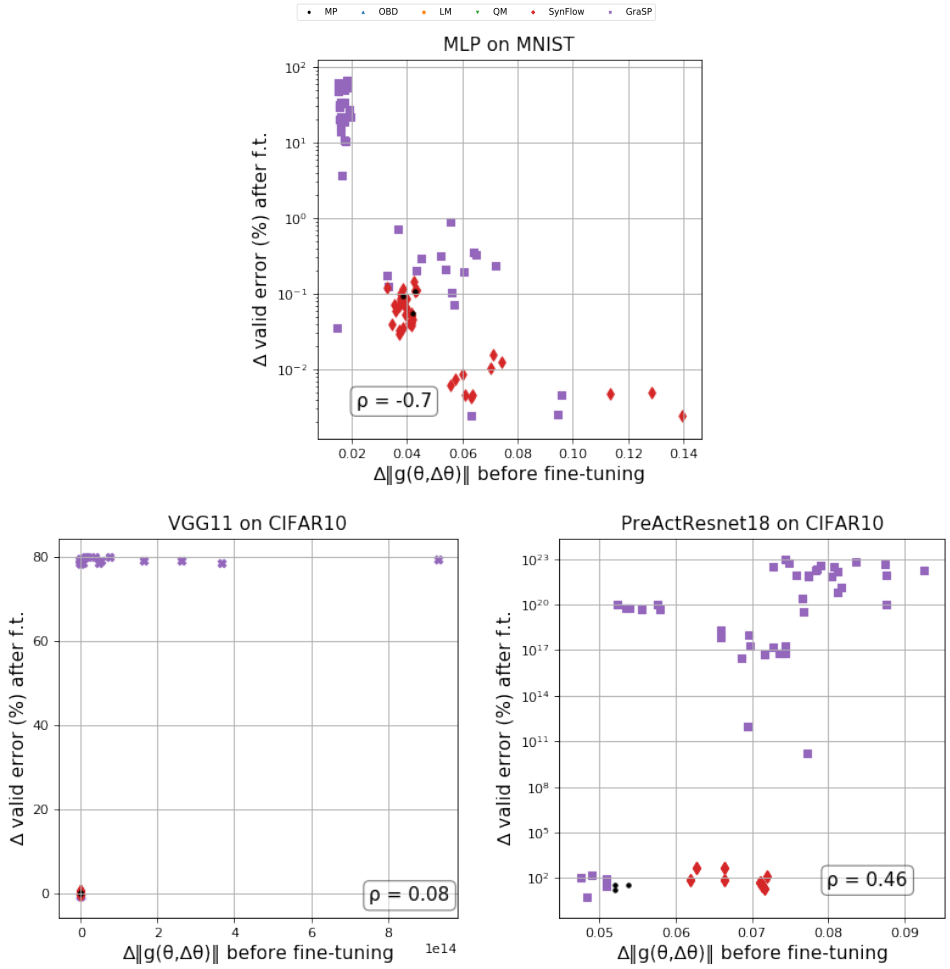


Figure 5.9: Gap of validation error after fine-tuning as a function of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$.

Performance gap Best preserving the original network function does not help produce better-pruned networks. Table 5.5 summarises the best validation error gap between the original networks and the pruned networks *after fine-tuning* for all considered criteria, loss-modelling and gradient-based. Amongst all configurations tested (π and λ), all criteria produce pruned-network that are in close range in

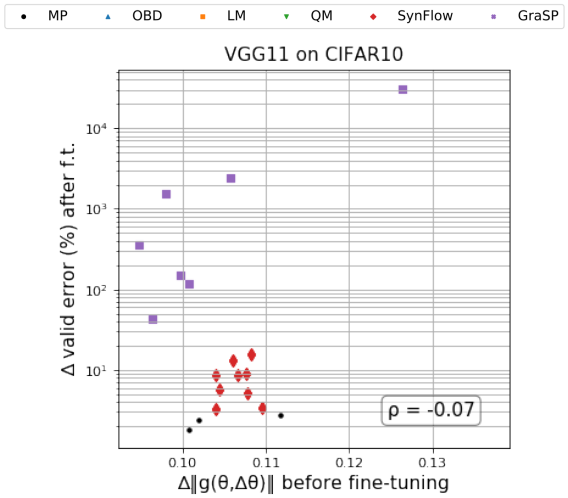


Figure 5.10: Gap of validation error after fine-tuning as a function of $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$. Zoomed in view of Figure 5.9 (center) for gradient-based criteria on VGG11. When we ignore points corresponding to pruned-networks that were unable to be re-trained (validation error 0.9), we do not observe any significant correlations.

Network	**Gap of Validation Error (%)**					
	MP	**OBD**	**LM**	**QM**	**GraSP**	**SynFlow**
MLP	2.4 ± 0.3	2.0 ± 0.1	1.9 ± 0.3	1.9 ± 0.2	**0.1 ± 0.1**	2.1 ± 0.2
VGG11	0.2 ± 0.2	-0.1 ± 0.2	**-0.3 ± 0.1**	-0.1 ± 0.1	**-0.5 ± 0.5**	0.3 ± 0.3
PreActResNet18	0.2 ± 0.2	0.2 ± 0.2	**0.1 ± 0.1**	0.2 ± 0.2	0.6 ± 0.2	1.5 ± 0.2

Table 5.5: Best validation error gap **before/after fine-tuning** the pruned networks (lower is better), for different pruning criteria, across values of λ and π. GraSP and LM are among the best performing criteria, but all the criteria validation error gap lie in a close neighborhood after fine-tuning.

terms of performance accuracy. As a reference, global random pruning resulted in validation error rate of 47.18 ± 6.8 % for the MLP and resulted in non-retrainable networks on CIFAR10 (with 90 % error rate). An important observation is that the hyper-parameters λ and π that give the best performing criteria in terms of $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ in Table 5.1, or Table 5.3, are not the same as the ones that give the best performing criteria after fine-tuning in Table 5.5.

Unstructured pruning produces good performing sparse model, however, the good accuracy of pruned models cannot be linked to the pruning objective alone. As a result, all pruning criteria are performing equally good despite preserving different aspects of the dense network.

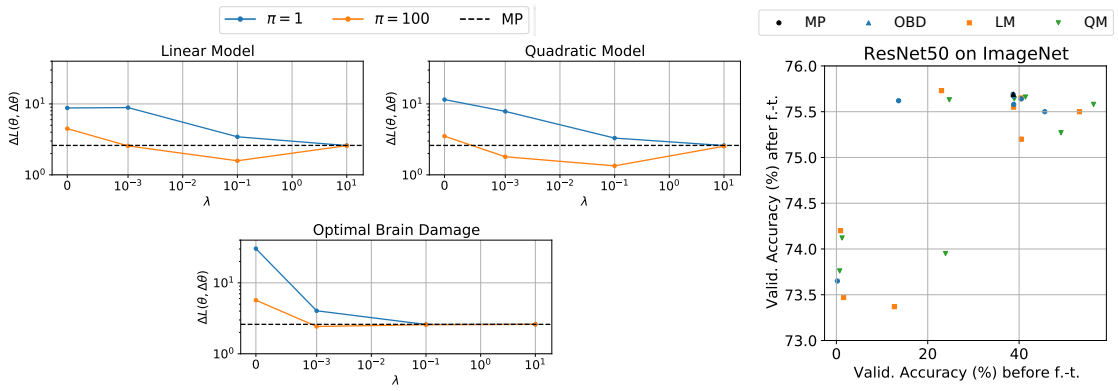


Figure 5.11: Same as Figure 5.3 and Figure 5.5, for the ResNet50 on ImageNet, with a sparsity of 70 %. Increasing the number of pruning stages and constraining the step size reduce $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. However, the best-loss preserving criteria, which maximize the validation accuracy right after pruning, do not produce better networks after fine-tuning. They perform similarly if their validation accuracy after pruning is $> 20\%$. Criteria that outperform MP right after pruning do not achieve better performance after fine-tuning.

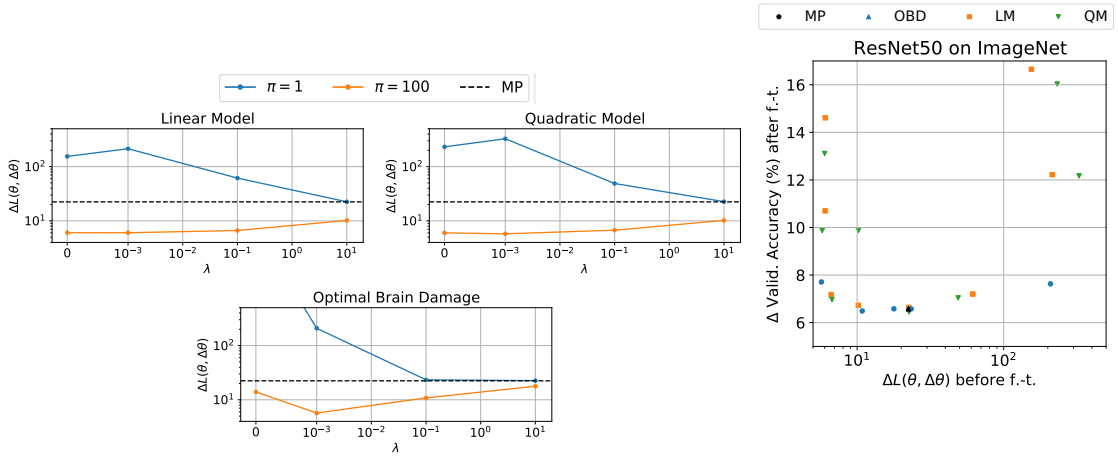


Figure 5.12: Same as Figure 5.11, but with 90 % sparsity. Increasing the number of pruning stages and constraining the step size reduces $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$. However, the best-loss preserving criteria, which maximize the validation accuracy right after pruning, do not produce better networks after fine-tuning.

5.4.3 Scaling to ImageNet

To investigate whether previous observations scale to larger datasets, similar experiments are conducted with loss-modelling criteria only due to compute limitations – LM, QM and OBD – on the ResNet50 on ImageNet. Before pruning, the network reached 76.41 % validation accuracy. Figure 5.11 presents results at 70 % sparsity, in a similar fashion as Figure 5.3 and Figure 5.5. Similar trends are observed: The best loss-preserving models are not necessarily the best models after fine-tuning.

When exploring a higher pruning ratio, the lack of relationship between best minimising the error gap *before/after pruning* and *before/after fine-tuning* is even more pronounced. Figure 5.12 is the same as Figure 5.11, but with 90 % sparsity. At that sparsity level, the validation accuracy right after pruning is close to random for all the pruning criteria. There is however quite a big variation in performances after fine-tuning: at equal performance before fine-tuning, some models achieve 70 % validation accuracy after fine-tuning, while others only reach 60 %.

5.4.4 Improving GraSP criteria

Previous experiments have shown some instabilities in GraSP criterion, especially its tendency to produce pruned networks that do not re-train properly due to high gradients. In their paper Frankle et al. (2021) showed that removing parameters with the lowest or highest GraSP saliency was producing very similar results, and thus proposed to prune weights with the lowest *magnitude* of GraSP saliency score: $s_k^{\|\text{GraSP}\|} = |\boldsymbol{\theta} \odot \mathbf{H}(\boldsymbol{\theta})\mathbf{g}|$. Figure 5.13 displays the $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ for different regularisation π and λ for $\|\text{GraSP}\|$ pruning variant. The effect of regularisation is much more visible, as a good tuning of the number of pruning stages π and norm regularisation λ is capable to constraint the gradient flow of the pruned network in a much more reasonable range compare to GraSP (Figure 5.7). Those observations reinforce the necessity to apply local steps, and more specifically multi-stage pruning when reducing the size of a deep learning model to better carry out the criterion objective.

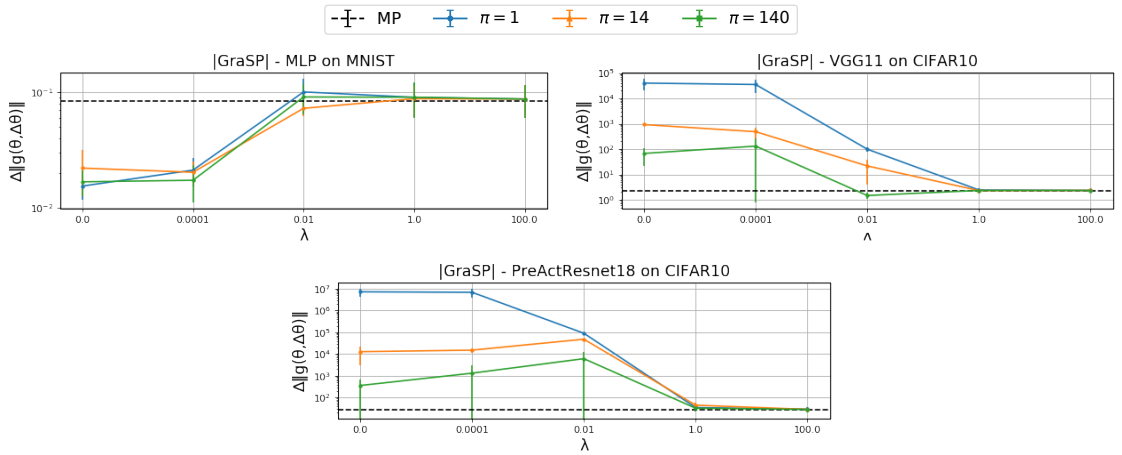


Figure 5.13: Same at Figure 5.7, $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ for different number of pruning stages π, as a function of λ for $\|\text{GraSP}\|$ pruning criteria. We observe that removing the lowest magnitude of GraSP, in addition to regularisation λ and multi stage pruning π lead to better performance than the original GraSP heuristic. We are able to prevent the gradient norm to being too large.

5.5 Chapter Summary & Discussion

5.5.1 Problem Statement

Pruning consists of removing parameters from a deep neural network to reduce the size of the model. For that, pruning criteria are used to estimate the importance of every single parameter and decide whether they are to be kept or discarded. The importance of a parameter is measured based on its influence over some pruning objective that usually aims at preserving certain properties of the original network. Three common objectives in unstructured pruning are: preserving the loss of the original model to preserve the learning trajectory (LM, QM, OBD), monitoring the signal propagation through gradient flow to ensure good re-trainability (GraSP, SynFlow), or simply removing the least salient connection preserving the norm of the weights (MP).

Despite the wide variety of pruning criteria, a criterion that consistently achieves state-of-the-art performance has yet to be found. But to design the ultimate pruning criterion it is fundamental to first assess and understand what components in existing pruning criteria and objectives contribute to producing better-pruned models. Evaluating the integrity of pruning criterion design is the first step toward more

efficient pruning.

When pruning a deep neural network, practitioners ultimately care about the end accuracy after fine-tuning. In addition to the removal of parameters, pruning is almost always followed by a fine-tuning phase during which the pruned network recovers for the loss in performance accuracy. Thus, assessing the efficiency of a pruning criterion can be done by checking how much does preserving the original network properties (pruning objective) correlate with producing better networks (end-accuracy). In summary:

1. Before fine-tuning: evaluating how good are each criterion at solving their pruning objective. In other words, are LM, QM, OBD criteria good at preserving the loss, and GraSP, SynFlow good at preserving the gradient flow between the original and pruned network?

2. After fine-tuning: Assessing if better preserving the original network functions (loss or gradient) actually lead to better performing pruned models.

To obtain the best possible pruned model, two tricks – multi-stage pruning π, and norm regularisation λ – to enforce various degrees of locality enforcement were explored. The correlation between preservation performance and prediction accuracy was studied in a second time.

5.5.2 Discussion & Conclusion

To a certain extent, pruning criteria are good at preserving the network functions. On small network architectures, such as MLP models, preserving the loss or gradient flow is a good indicator of performance after fine-tuning. But on larger network architecture, VGG11 or PreActResnet18, no significant link between preservation capabilities and end-goal accuracy were observed. It is worth noting that for these larger networks, preserving the gradient flow seemed to be a slightly better indicator than the loss, as monitoring the gradient norm can help better detect damages in the sparse architecture that would prevent re-training.

The architecture has an important influence on the performance of pruning methods as results observed on small networks do not scale to larger ones. This chapter suggests that there are no optimal universal pruning criteria. There is no clear evidence that better preserving the network function leads to better fine-tuning performance. Even the best-motivated pruning criteria, fail to consistently produce good pruned networks. Additionally, a simple magnitude heuristic, such as MP, obtains really good performance in practice despite its simplicity. This raises concerns about the core foundation and adequacy of current pruning criteria design that aim to provide efficient pruned networks after fine-tuning, but often lack consideration for the correlation between pruning objectives and final performance after fine-tuning.

To improve upon existing criteria, deeper investigation into pruning criteria are required to understand what properties of modern architectures make pruning crtieria to fail at producing reliable sparse models. Is it caused by larger networks not being trained to convergence? Is it the depth of the architecture inducing too many non-linearity? Or simply, is the prediction accuracy not representative enough of the granularity between pruned networks. What are we missing? Those questions will be explored in the next chapter, Chapter 6.

Chapter 6

Beyond accuracy: evaluating pruning performance

The growing interest behind network pruning has been motivated by the need for smaller, more compact, deep learning models for real-world applications. But it was also reinforced by the potential of pruning to help understand the benefits of parameterisation in deep neural networks in order to design more efficient architectures and training procedures from scratch. Over the years, many pruning methods and frameworks have been developed exploring different aspects of deep neural networks. In the previous chapter (Chapter 5) different families of unstructured pruning criteria were presented with the intent of creating sparse architecture that preserves characteristics from the original network (loss or gradient flow) to reach similar performances. Despite the wide variety of pruning approaches, no clear gain could be observed from best preserving the dense network loss or gradient flow. This research suggests that in their current state pruning criteria are not fully proficient. A statement supported by other recent works showing the instability of pruning at initialisation (Bartoldson et al., 2020; Frankle et al., 2021).

Many factors can influence pruning performance outside the pruning criterion: the network architecture, re-training strategy, complexity of the dataset. For instance, Renda et al. (2020); Blalock et al. (2020) showed the importance of re-

training strategy, especially the choice of hyper-parameters during the fine-tuning phase of the pruned network. Follow-up research by Le & Hua (2021) confirmed the importance of the learning rate. Other works have focused on studying the flexibility of the masks produced by pruning (Morcos et al., 2019; Zhou et al., 2019; Paganini & Forde, 2020). But in all these studies, it is often the best accuracy after fine-tuning the pruned model that is used to ultimately measure the performance of the sparse model.

In this chapter, we conduct exploratory research to study the impact of pruning beyond solely looking at performance accuracy. Two specific aspects of deep neural networks are investigated: the role of parameterisation in pruning performances – the model architecture, and the robustness and fairness in the pruned model predictions. Section 6.1 present a study on the impact of parameterisation on the soundness of pruning criteria. We iterate over the results presented in Chapter 5 to explore when does best preserving the original network's functions (loss and gradient-flow) stop being a good pruning objective. Section 6.2 will present and review an interesting property of sparse neural networks at finding implicit biases from the training data demonstrated by Hooker et al. (2021). We will extend this work to study the behaviour of the different unstructured pruning criteria presented in Chapter 5.

6.1 On the impact of parameterisation

In chapter 5, different pruning criteria were analysed to see if better preserving the original network function (loss and gradient flow) lead to better performance after fine-tuning. It appeared that architecture plays an important role in the outcome of the pruning criteria. On smaller architectures (MLP), preserving the loss or gradient flow was a good indicator for predicting the accuracy performance of the pruned model after fine-tuning. Best preserving the loss and improving signal propagation in the sparse model led to higher accuracy performance after fine-tuning. However, those observations failed to scale to larger models (VGG11 and PreActResNet18)

where no correlation could be observed between better preserving the original network loss or gradient flow, and the performance of the pruned model. Therefore, a natural question to ask is: *what are the differences between the architecture tested previously*?

Modern architectures (VGG - Resnet-*like*[1]) have a higher number of parameters but they are also deeper as they contain many more layers compared to small MLP architectures. Thus, two factors may cause the correlation between the preservation abilities and the end-performances of the pruned model to drop: (1) the overall number of parameters, or (2) the depth of the architecture. Deeper architectures can be harder to optimise and usually require more regularisation as the signal encounters more non-linear activation units. While augmenting the number of parameters implies more computations making it more difficult to approximate the loss or gradient flow, and therefore removing the right set of parameters.

To estimate whether the decrease in pruning proficiency is due to a large number of parameters or the network architecture in itself (the depth), we conduct a study to compare the behaviour of *wide* – parameters added increasing the width of the network – versus *deep* – increase in parameters by adding more layer – networks. Two types of building blocs architecture are tested – fully connected and convolution – to further investigate how the inner complexity in the network impact the performances. Note that for the case of this study, a pruning criterion is considered proficient if better preserving the original network function correlates with good performances after fine-tuning of the pruned network. We do not care about the best accuracy after fine-tuning.

6.1.1 Methodology: Wide verses Deep network

Setup The effect of increasing the number of parameters is tested on both a Multi-Layer Perceptron (MLP) model architecture and a Convolution Network (Convnet) architecture. The number of parameters is scaled up by increasing the number of

[1]Resnet-like architecture englobe Resnet and PreActResnet architectures

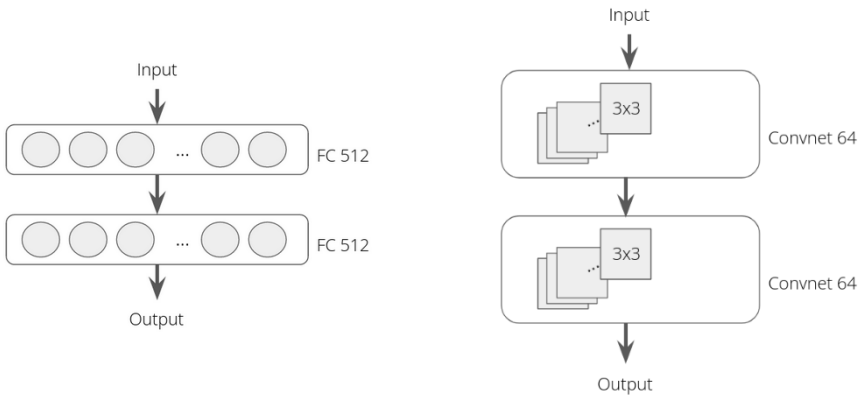
neurons, or respectively filter for the Convnet, either adding new units on existing layers (width) or creating new layers (depth). The baseline architecture for both models consists of 2 layers with 512 units for the MLP, and 64 filters for the Convnet. The number of neurons is then scaled-up by a factor of $\alpha \in \{2, 4, 6, 8\}$ for the MLP and $\alpha \in \{2, 4, 6\}$. Figure 6.1 summarises the different model architectures studied.

It is worth noting that because wider networks share more connections, it results in a higher overall number of parameters despite having the same number of neurons, or filters. For fair comparisons of the pruned models, and to ensure that better performances are not due to a higher number of active parameters, we constrain the number of parameters remaining after pruning for both networks wide (*width scaling*) and deep (*depth scaling*) to be the same. In practice, this means that for a given pruning ratio, the *wide* models will be pruned more aggressively while maintaining the same number of active parameters than the *deep* models.

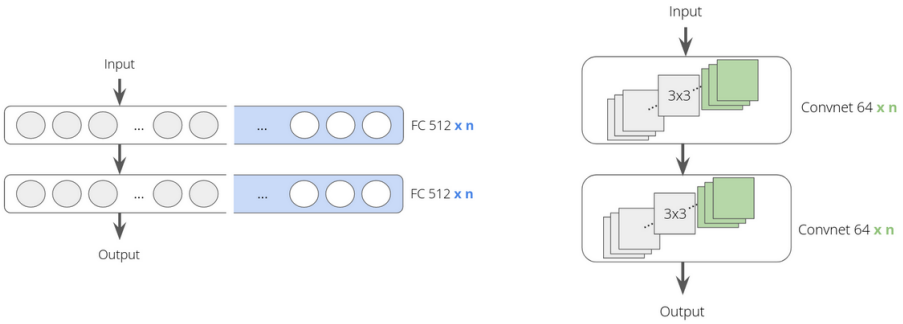
Evaluation Performances are evaluated on both MNIST and CIFAR-10 dataset, with pruning ratio of 98.8% and respectively 95.6%. More details about the training hyper-parameters can be found in Appendix C.1.1. Performances are evaluated computing the *Kendall* rank correlation between the best validation error after fine-tuning, and the preservation abilities $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ (loss), and $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ (gradient flow).

$$\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)} \tag{6.1}$$

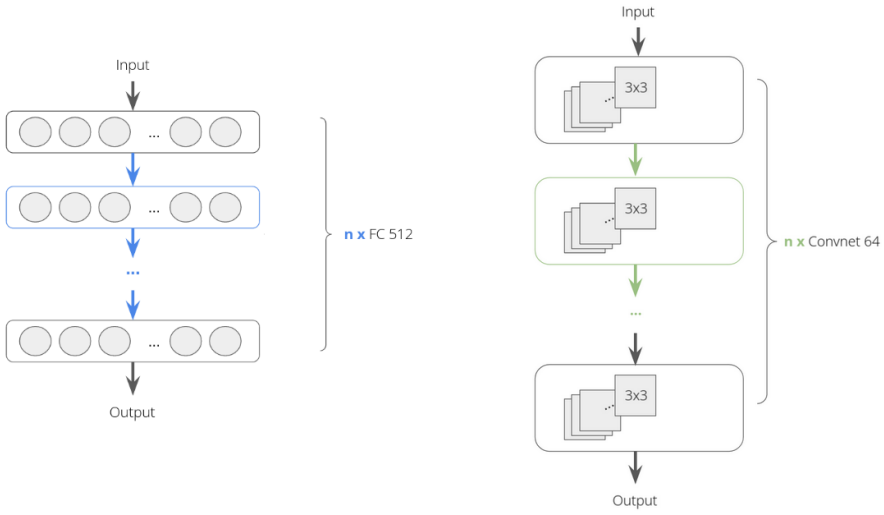
where n_c and n_d are respectively the number of concordant and discordant pairs, and n the total number of data points. Note that in Chapter 5 the *Spearman* rank correlation ρ was used, here Kendall scores τ is preferred as our sample size is smaller, 30 data points versus 75 to 410 data points in Chapter 5 where many locality variants were considered for the different pruning criteria. Here the regularisation factor λ is fixed to 0 and the number of pruning iterations π to $\{1, 100\}$. While the two rank correlation metrics might lead to slightly different measurements, the overall inference is unchanged. The following pruning methods are considered: LM



(a) *Baseline* model architectures with 2-layers, MLP (left) and Convnet (right).



(b) *Width scaling* model architectures, adding neurons on existing layers. MLP (left) and Convnet (right).



(c) *Depth scaling* model architectures, adding neurons on new layers. MLP (left) and Convnet (right).

Figure 6.1: Different model architectures considered for studying the impact of parameterisation over pruning criteria integrity, MLP (left) and Convnet (right). Every layer is followed by ReLU activations, and a Softmax at the end for the classifier.

with $\pi \in \{1, 100\}$; QM, GraSP-abs, SynFlow with $\pi = 100$; and MP with $\pi = 1$, where π is the number of pruning stages. See Chapter 5 for all the details regarding the different pruning criteria.

6.1.2 Experimental results & observations

6.1.2.1 Multi Layer Perceptron (MLP)

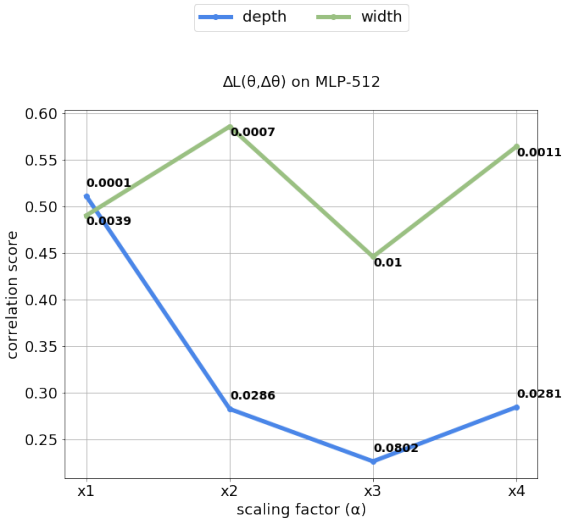
First, we study the impact of parameterisation over an MLP architecture. In Chapter 5 we saw that on a small MLP architecture[2] there was a correlation between best preserving the network properties and the performance of the pruned model after fine-tuning. While the correlation was positive for preserving the loss $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ ($\rho = 0.67$), it was negatively correlated for preserving the gradient flow $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ (($\rho = -0.7$)). Thus we ask two questions:

- *Will the correlation drop as the number of parameters in the MLP increase*?
- *Does the number of layers impacts the performance of pruning objectives*?

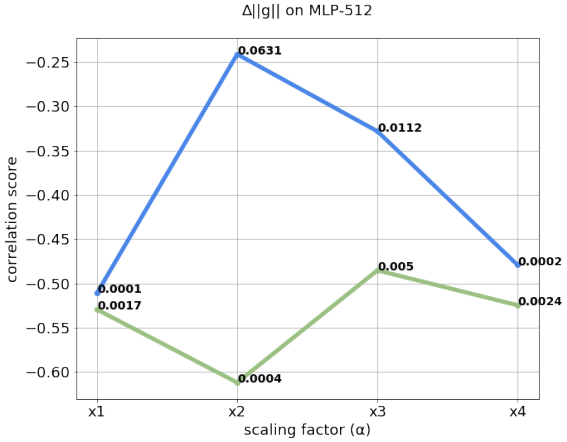
Figure 6.2 displays the correlation score between the preservation abilities and the validation accuracy after fine-tuning the pruned model (y-axis) as we increase the number of neurons in the networks (x-axis). For loss preservation methods $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$, the correlation score remains more or less constant, oscillating between 0.45 and 0.58 as we increase the number of parameters on existing layers. However, when increasing the number of parameters by adding new layers, the correlation scores drop significantly (variation of -0.3).

When considering gradient flow preservation abilities, similar tendencies can be observed. Augmenting the width of the network only cause a little decrease in correlation score – average deviation of $\sigma = 0.05$, while results observed on deeper networks observe a larger variation ($\sigma = 0.11$). On a very deep network (blue line – $\alpha = 8$), similar performances as the baseline model ($\alpha = 1$) are observed. It appears that sparse networks that augment the most the gradient flow get better performance

[2]MLP with 3 layers with respectively 300, 300 and 100 units



(a) $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ correlation



(b) $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ correlation

Figure 6.2: Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a MLP with 2 layers with 512 neurons on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored

after fine-tuning, thus gradient flow is a good indicator for predicting accuracy performances. However, the good performance of gradient flow correlation might be a result of the presence of *outliers* caused by the pruned networks that failed to re-train properly. Tables C.1 and C.2 in Appendix C summarise the different accuracy score obtained for the different wide and deep MLP networks – scaled either adding new neurons on existing layers or by adding new layers. For network scaled adding new layers (Table C.2), some pruning methods failed to produce fine-tunable models (large accuracy errors). Deeper networks are indeed more at risk of layer collapsing

when the architecture is damaged beyond recovery. Such damaged networks result in outliers that can greatly influence the correlation score (see Figure 6.3).

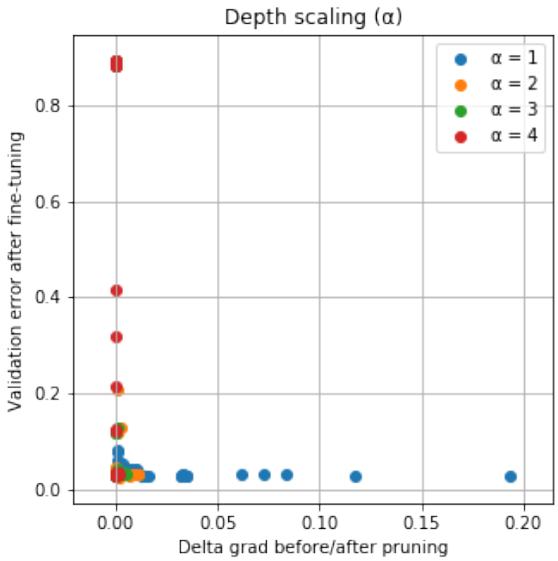
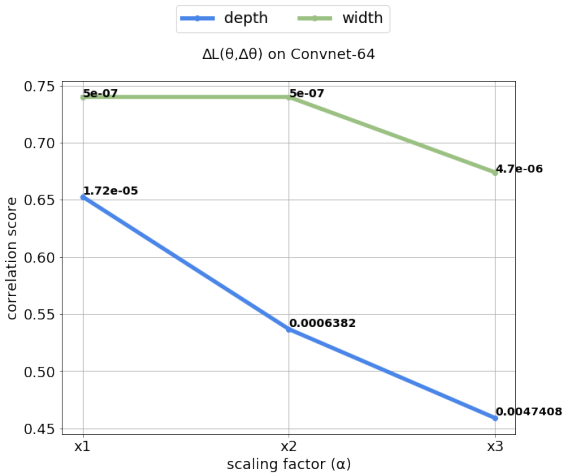


Figure 6.3: Scatter plot displaying the $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ *before/after pruning* as a function of the best validation error obtained after fine-tuning (where α indicates the scaling factor). We can observe that there is a range of networks for $\alpha = 4$ (MLP with 8 layers) where the sparse model failed to retrain accurately leading to high error accuracy for low change in gradient flow.

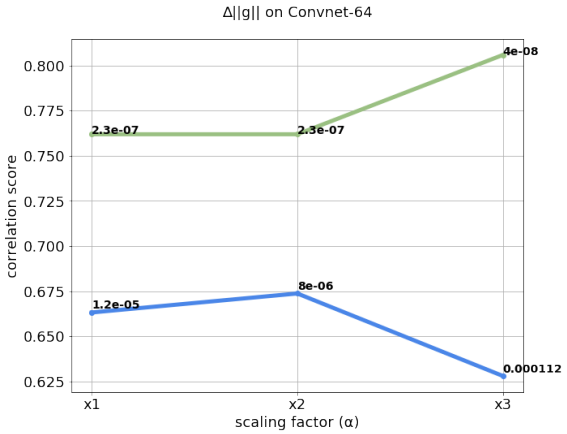
Therefore, on MLP architectures the number of parameters in itself is not what is causing the correlation to drop but rather the depth of the network. A possible explanation is that adding more layers introduces more non-linearity making the networks harder to optimise. Thus approximating the loss or gradient flow becomes harder and less reliable. We further investigate whether those observations hold on more complex neural networks, for Convnet architectures.

6.1.2.2 Convolutional Neural Networks on CIFAR-10

We first consider a set of Convnet models trained on CIFAR-10 and pruned at a 95.6% ratio. Figure 6.4 displays the correlation scores as a function of the scaling factor for both *loss preservation* (top) and *gradient-flow preservation* (bottom). For a simple two-layers architecture with 64 filters on each layer ($\alpha = 1$), best preserving the original network function, loss or gradient flow, seems to be a good indicator for the best accuracy of the pruned model ($\tau = 0.65$ and $\tau = 0.74$ respectively).



(a) $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ correlation

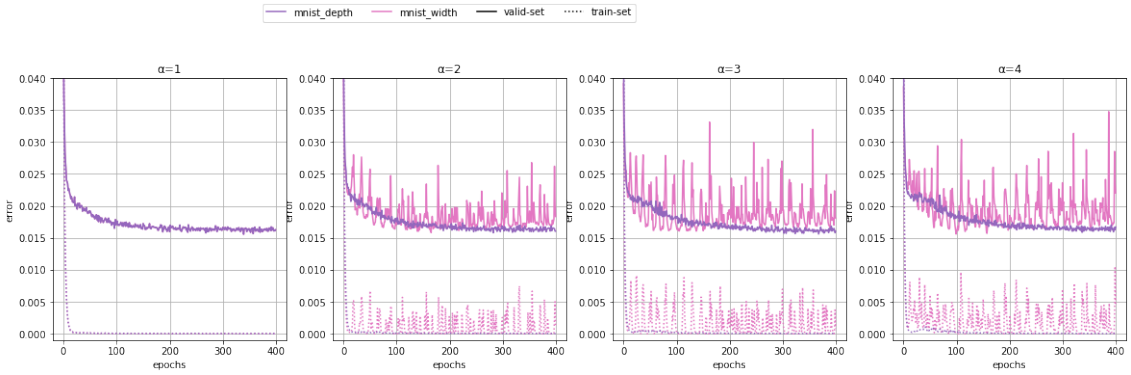


(b) $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ correlation

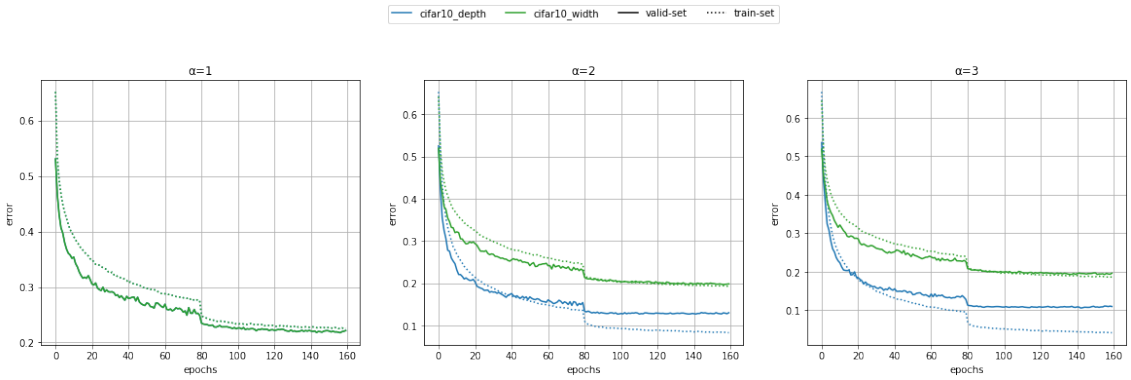
Figure 6.4: Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a Convnet trained on CIFAR-10 with 2 layers with 64 filters on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored.

Augmenting the number of parameters by adding more filters either on existing layers (width) or by adding new layers (depth), does not impact the correlation much meaning that no matter the number of parameters, best preserving the original network function is a good indicator for the end performance of the pruned model. However, for loss-preservation, we can observe a drop in correlation (-0.2) when increasing the number of parameters, especially for deeper architectures.

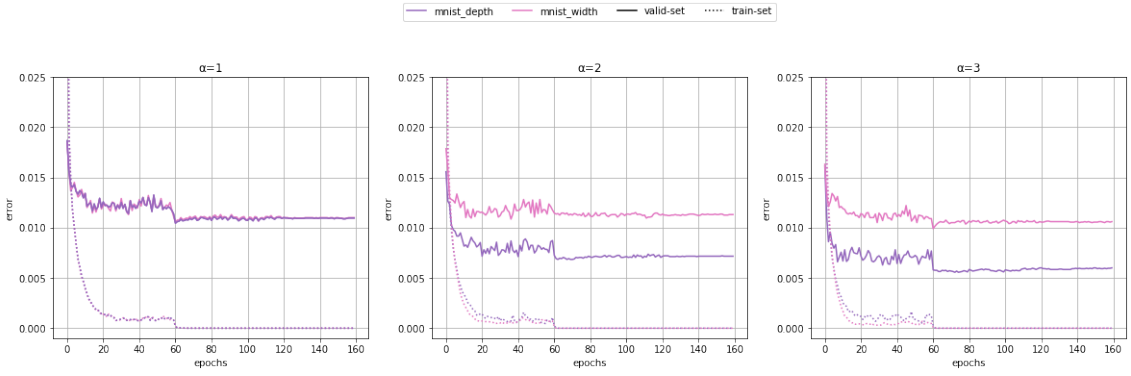
The impact of parameterisation on correlation metrics is much less pronounced on the Convnet architectures than on the MLP architectures. One possible explanation is that the different Convnet models are not solving the task properly.



(a) MLP on MNIST



(b) Convnet on CIFAR-10



(c) Convnet on MNIST

Figure 6.5: Training curves for different baseline model (dense architecture) for MLP (*top*) and Convnet on CIFAR-10 (*middle*) and Convnet on MNIST (*bottom*). For each, we display the baseline for different scaling factors α.

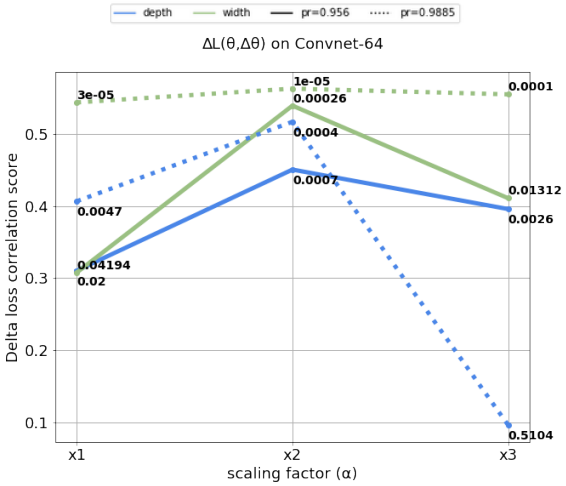
Figure 6.5b displays the training classification error curves for the different Convnet baselines trained on CIFAR10 for the training (dotted) and validation (plain) set. We can observe that wide networks (green line) slightly *under-fit* CIFAR-10 dataset with an average miss-classification error rate between 20 % and 25 %. On the other hand, the deeper networks (blue line) are better at solving the task – 10 % error for 6 layers and 15 % for 4 layers – but have a tendency to *over-fit* with the training and validation curves diverging from one another.

From previous results on MLP and Convnet trained on MNIST and CIFAR-10, we observed that as the network gets deeper the correlation tend to drop. We hypothesised that it was due to an increase in non-linearity making the training harder from an optimisation perspective. Figure 6.5b brings a new insight suggesting deeper networks are at higher risk of over-fitting. If wider networks under-fit the dataset, deeper networks over-fit, *is best preserving the network function a good indicator for the accuracy of the pruned network only when the network is not over-fitting?*

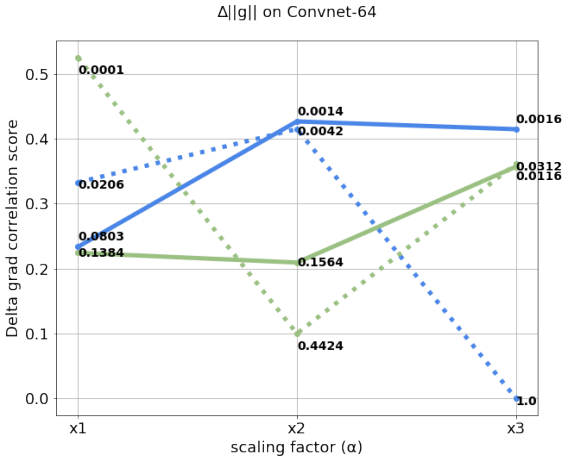
6.1.2.3 Convolutional Neural Networks on MNIST

To assess the impact of *under* and *over-fitting* on the correlation factor, we now investigate the behaviour of Convnets on MNIST dataset where both, wide and deep networks over-fit the training data – (see Figure 6.5c). Two pruning ratio are considered: $pr = 95.6\%$ for consistency with the previous experiments on CIFAR-10, and $pr = 98.85\%$ to prune the model at its best potential as MNIST – a grey-scale collection of digit numbers – is far less complex than CIFAR-10 – a RGB collection of 64×64 images. Results are presented in Figure 6.6.

When trained on MNIST considering a pruning ratio of 95.6% (plain lines), augmenting the number of parameters can help to improve slightly the correlation score. We cannot conclude much from increasing the width of the network (green plain line) as most correlation points observed are not statistically significant ($p - value > 0.01$). Also, it is worth noting that significant correlation scores observed



(a) $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ correlation



(b) $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ correlation

Figure 6.6: Correlation score as a function of the network width (green line) or depth (blue line). The baseline model (x1) consists of a Convnet trained on MNIST with 2 layers with 64 filters on each layer. Numbers displayed indicate the *p-value* scores. When a p-value score is >0.01 the associated measurement is not considered to be statistically significant and can be ignored.

are far less pronounced than previous experiments on CIFAR-10, with correlation scores between 0.3 and 0.5 compare to 0.75 observed earlier.

When considering a higher pruning ratio $pr = 98.85\%$ (dotted line) we obtain slightly better results where measurement are significant ($p - value$<0.01).

Note that compared to MLP, the correlation between gradient-flow preservation and end accuracy is positive. Convnets are trained for less epochs than the MLP models and use early stopping meaning that the training is stopped before it has reached full convergence. Thus, preserving the gradient flow might help reach better

training accuracy compared to a fully converged network – as it is the case with the MLP – where preserving the gradient flow corresponds to preserving zero-gradient values.

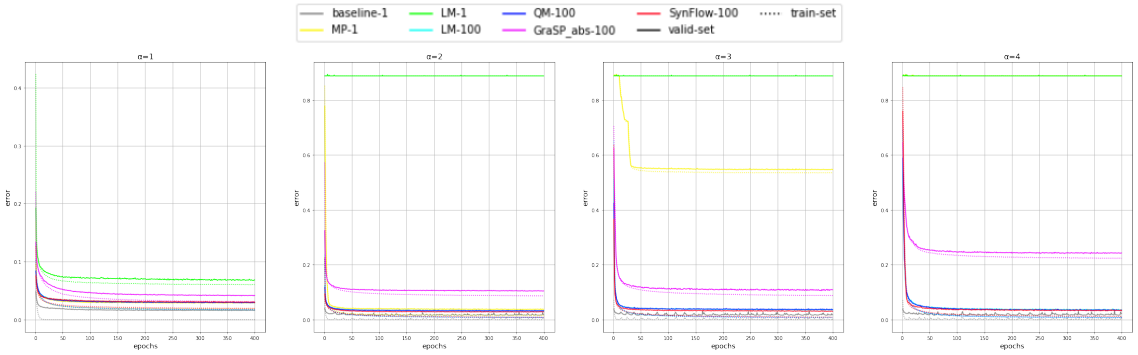
Overall, results observed on Convnets trained on MNIST are less significant than previous experiments on Convnets on CIFAR-10 or MLP on MNIST. But Convnets trained on MNIST over-fit more than their counter-part trained on CIFAR-10, supporting our hypothesis that over-fitting impair the performance of pruning criterion.

6.1.2.4 Fine-tunining

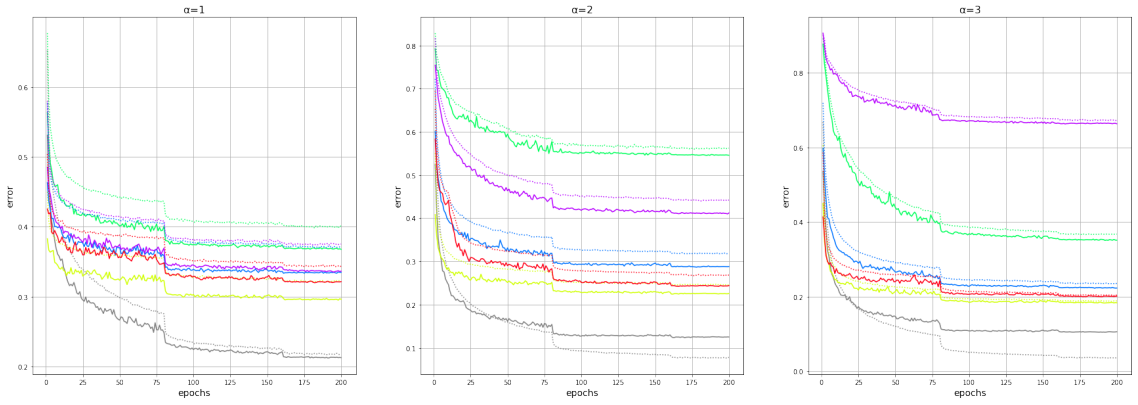
When the network over-fit during the initial training phase, as for the MLPs and Convnets trained on MNIST (see Figure 6.5), best preserving the original network function do not constitute a consistent good indicator to predict the pruned model performances with absolute correlation score varying from 0.25 to 0.55 (see Figure 6.2 and Figure 6.6). To investigate whether over-fitting causes optimisation issues when fine-tuning the model, we investigate the fine-tuning phase.

Figure 6.7 displays the fine-tuning curves (validation and training error) for the different models' architectures and datasets tested when scaled up adding new layers (depth). For network scaled by adding new units on existing layers (width), fine-tuning curves are presented in Appendix C Figure C.2. Networks over-fitting during the training phase, tend to produce pruned networks that will over-fit during fine-tuning. This can be observed on Convnets on MNIST pruned at 95.6% (Figure 6.7c) and MLP on MNIST (Figure 6.7a). By augmenting the pruning ratio – Convnet on MNIST pruned at 98.85% (Figure 6.7d) – the gap between the training and validation curves is shrunken but we observe a loss in accuracy and performance are not as good as when pruning at 95.6%.

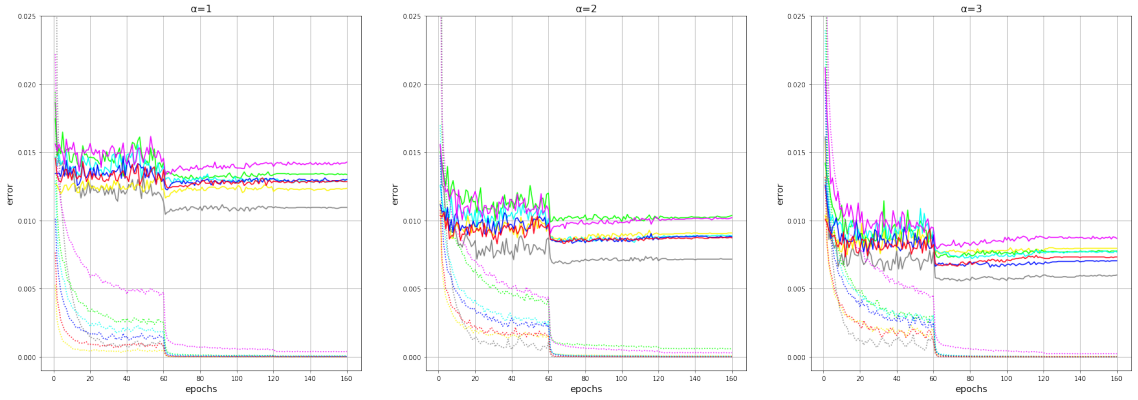
For Convnets trained on CIFAR-10 (Figure 6.7b) pruning the network generate sparse architectures that do not over-fit when fine-tuning. This could explain the good correlation score ($\tau = 0.75$) observed in Figure 6.4.



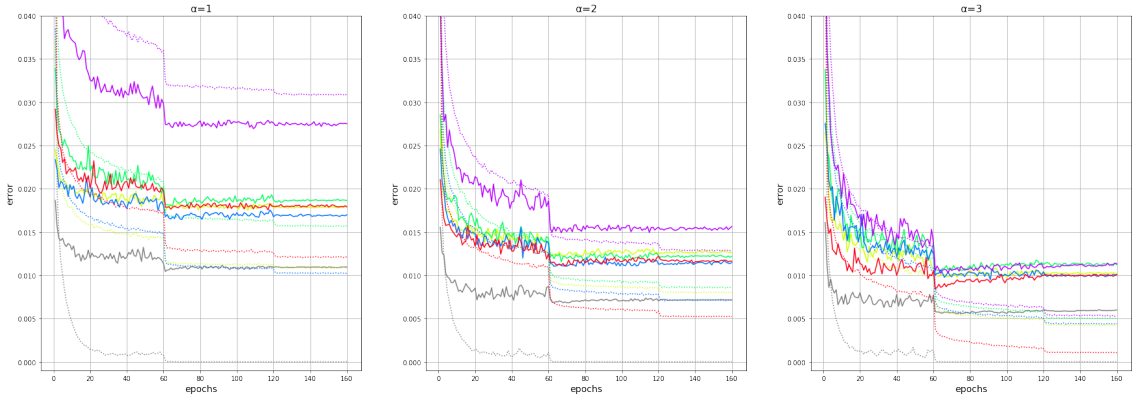
(a) MLPs on MNIST



(b) Convnets on CIFAR-10



(c) Convnets on MNIST with pr=95.6%

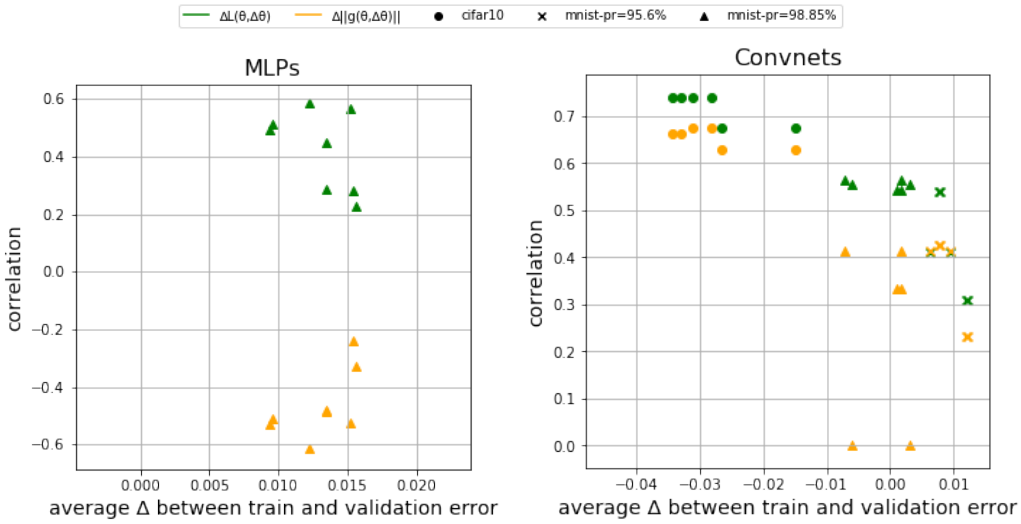


(d) Convnets on MNIST with pr=98.85%

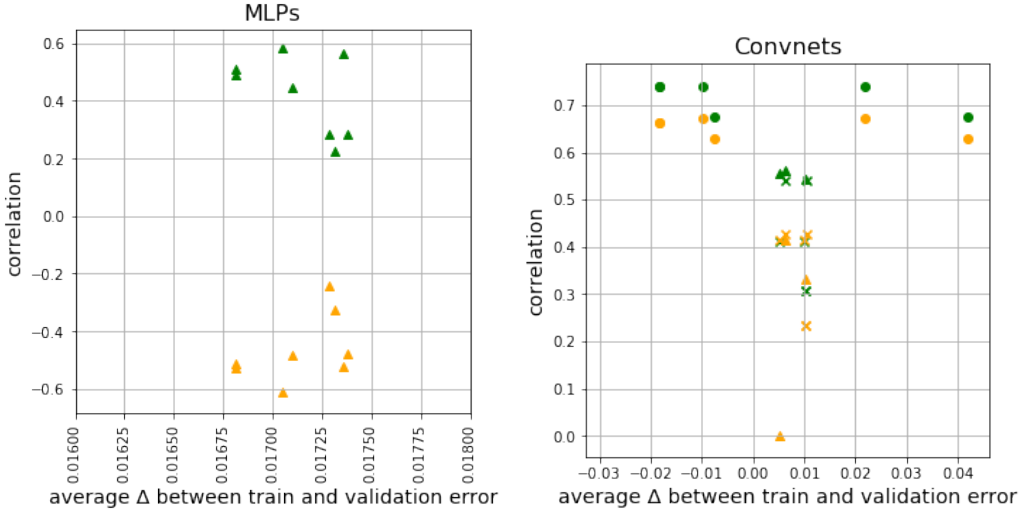
Figure 6.7: Fine-tuning curves (classification error) for different model architectures scaled on the depth. (See Figure C.1.3.2 in Appendix C for model architectures scaled on the width.)

Similar results can be observed for networks scaled on the width (see Appendix C, Section C.1.3.2).

To confirm whether over-fitting influences if best preserving the network function is a good indicator for ensuring good performances of the pruned model, Figure 6.8 displays a scatter plot of the correlation scores (both $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ and $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$) as a function of the mean divergence between the training and validation curves averaged over all pruning methods.



(a) During fine-tuning (over-fitting of sparse models)



(b) During initial training (over-fitting of dense models)

Figure 6.8: Scatter plot of the average divergence between the training and validation curves verses the correlation score between best preserving the original network function ($\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ and $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$) and end-accuracy of the pruned model.

When considering the fine-tuning phase, ie. *sparse* architectures (Figure 6.8a),

for both Convnets models (right) and MLPs models (left), the lesser the divergence between the training and validation curves, the better the correlation score. However, when only considering the initial training phase, ie. *dense* architectures, (Figure 6.8b), the relationship between the correlation score and over-fitting is much less pronounced.

We can conclude that preserving the original network functions – loss $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ or gradient-flow $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ – is a good candidate to select unimportant parameters, but because these statistics (loss and gradient-flow) are computed on the training set, the more the network over-fit, the lesser those will be relevant to produce good pruned networks.

6.1.3 Discussion and Limitations

6.1.3.1 Discussion

While the initial motive behind pruning is usually to reduce the number of parameters and thus the size of a deep neural network, pruning can also be seen as a mechanism that injects noise in the model to better generalised to unseen data. Bartoldson et al. (2020) demonstrated empirically that the more instability a pruning method injects, the less sensitive it is to data changes, thus the better the generalisation performances. By preserving the original network function, we produce more stable pruning criteria that might have a tendency to be too reliant on the training data (less noisy). This could explain why despite its simplicity, Magnitude Pruning criterion remains highly competitive and with the right set of hyper-parameters often outperform other criteria more mathematically grounded. Also by removing natural noise, pruning criteria (LM, QM, GraSP) might encourage over-fitting. A possible way of exploring this hypothesis would be to investigate the flatness of the pruned model by looking at the trace of the Hessian as performed in Bartoldson et al. (2020).

Over-fitting during fine-tuning can also indicate an optimisation problem where another choice of hyperparameters might be more appropriate. Hyper-parameters,

especially the choice of the learning rate, plays a crucial role in the performances in terms of accuracy of the pruned model as demonstrated in Le & Hua (2021) and Renda et al. (2020). However, because the network is over-fitting during the training of the dense network, it is unlikely that solely performing hyper-parameters search will solve this issue. More regularisation is required to enhance training of sparse architectures.

Lastly, picking the right degree of sparsity within a deep network is often a question of trade-off between compute time and accuracy performance (see Chapter 4 Section 4.1.4). A higher ratio of pruning can help mitigate the gap between training and validation but often result in a loss in accuracy as demonstrated with Convnets trained on MNIST and pruned at 95.6% and 98.85%.

6.1.3.2 Limitations: Drop in accuracy & Layer collapsing

To find the optimal pruning ratio, the one that preserves the best accuracy of the original model, the best practice is to prune the network gradually while fine-tuning the pruned network in between each stage. By doing so, if pruning degrades too much performance we just have to revert to the previous sparse model, and thus it is easier to find a desirable trade-off between sparsity and accuracy. But such an approach can be compute-intensive.

Figure 6.9 presents the best accuracy (solid lines) obtained after fine-tuning the pruned model for different pruning ratios. The divergence between the training and validation error is also displayed (dotted line). The impact of pruning ratios over the pruned model performances can be separated into three-phase as described in the sparsity survey from Hoefler et al. (2021). When the pruning ratio is small – (Phase I), pr<50%] – there is a small benefit to pruning. Increasing the pruning ratio slightly improves the accuracy but the benefits compute-wise are often negligible. By pruning over 50%, we enter the optimal sparsity regime for which a good trade-off between accuracy and compute complexity can be found. In this stage, the pruned model first get a boost in accuracy, sometimes outperforming the baseline, before

its performance starts to slowly decrease – (Phase II), pr $\in [50\%, 98\%]$. But further increasing the pruning ratio – (Phase III), pr>0.98 – quickly degrades the pruned model performances, it usually corresponds to a stage where the architecture starts to be damaged beyond recovery.

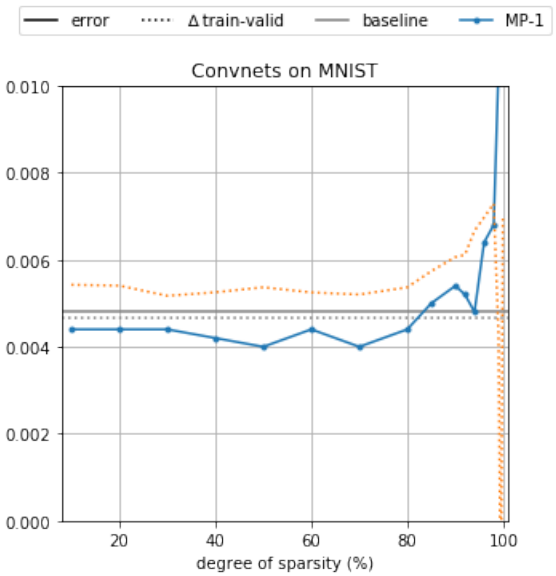


Figure 6.9: Classification error (plain line) as a function of the sparsity ratio for a Convnet trained on MNIST. The dense model accuracy is reported in grey. Dotted line display the average gap between the validation and training set error.

Also displayed in Figure 6.9 is the over-fitting factor during fine-tuning as a function of the pruning ratio (dotted line). In this context, over-fitting is measured as the average divergence between the train and validation curves. We can observe that pruning does reinforce over-fitting, however, there is a small regime during phase III where pruning can help mitigate over-fitting. However, this only happens when the network observes a significant loss in accuracy which is not desirable in practice.

Under Phase III the accuracy of the sparse models drops abruptly which indicates that the architecture of the neural network gets damaged beyond recovery. At high pruning ratios, it is harder to balance the number of remaining parameters between the different layers, some layers might suffer from a *collapsing* effect. *Layer-collapsing* is considered to be the major obstacle for pruning before training (Tanaka et al., 2020), but we show that this phenomenon also occurs when pruning during

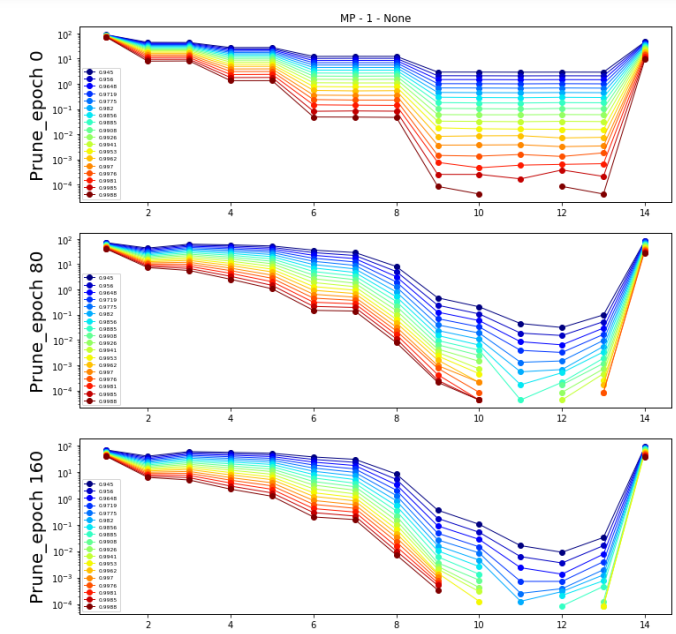


Figure 6.10: Ratio of parameters remaining per layer (x axis) for a VGG16 architecture when pruned before training (top), at the middle of training (middle) or after training (bottom). Magnitude pruning is applied in this example. Different colors correspond to different ratio of pruning, with blue being a small pruning ratio and red high pruning ratio.

or after training.

Figure 6.10 presents the sparsity per layer for different pruning ratios on VGG16 trained on CIFAR-10 dataset. We can observe that layer-collapsing occurs at different times during training.

Global pruning – when the sparsity ratio is applied across all layers rather than uniformly – is the preferred approach when pruning a deep neural network compare to layer-wise pruning – where we remove the same ratio of parameters on each layer. It offers more flexibility but also encourages layer collapsing. A simple trick to prevent layers from collapsing could be to enforce a minimum number of required parameters per layer.

The impact and importance of the scope of sparsity – *global* verses *layer-wise* – has not been thoughtfully studied in the literature. However, recent studies by Frankle et al. (2021) demonstrated that in the context of pruning at initialisation, the parameters' values did not matter but the layer sparsity did. Thus it would be interesting to study further if preventing layers from collapsing can boost the pruning

performances

Note that all the experiments presented in this section were conducted on MP. Key pruning ratio that defines the different pruning phase, or layer collapsing, are not invariant to the pruning method. But similar phenomenons can be observed. (See Appendix C.1.4)

6.2 On Fairness and Robustness

The main objective in research centred around pruning criteria is to design a heuristic to estimate the importance of parameters in order to reduce considerably the compute load while preserving the dense network accuracy. The aim is ultimately to find a criterion that manages to preserve the dense network accuracy under very high sparsity ratios. Thus, the effectiveness of a pruning criterion is usually assessed by measuring how far the network can be pruned (pruning ratio) while preserving a good accuracy, omitting other factors such as robustness to adversarial attacks, fairness in the pruned model predictions, or soundness of the importance heuristic in itself.

Despite their ability to reach similar prediction accuracy, dense and sparse models contain a radically different number of parameters. This raises concerns about the ability of validation accuracy to assess and evaluate the pruned model performances. Recent work by Hooker et al. (2021, 2020) demonstrated that while having similar top-line performance metrics, on a small subset of data, pruned models produce significant disparate outcomes compared to the original model. Unlike the belief that by removing parameters we produce more robust models, the authors showed the opposite: sparse models with high levels of sparsity reinforce existing bias in the datasets making them less reliable under adversarial attacks.

Whether it is at initialisation or end of training, it is hard to differentiate between unstructured pruning criteria, no criterion consistently out-perform the others. Frankle et al. (2021) showed that at initialisation all pruning methods are equal, while in 5 we demonstrated that all unstructured pruning criteria had similar performance.

In this section we explore the following questions:

- *Is assessing fairness in prediction a better tool to differentiate between different unstructured pruning criteria*?
- *Does pruning at initialisation generate better – fair and robust – sparse models than when pruning is applied after training as the network has not learned any prior?*

6.2.1 Methodology: Comparing fairness of pruning criteria

Pruning can be seen as a regularisation tool, or a way to reduce noise in the learning representations. Naively, one could think that by reducing the number of parameters the sparse model is forced to learn better general features which would lead to a boost in generalisation performance. If pruning was producing such fair and robust sparse models, it would:

- preserves similar degrees of per-class accuracy between the dense and sparse model
- be more robust to adversarial attacks. Small changes in the data points (injecting noise) should not impair the prediction

In this section, we focus on the fairness assessment of pruned algorithms considering different unstructured pruning criteria. Because pruning criteria rely on different importance measures, we seek to explore whether removing the parameters based on the loss or gradient-flow information or simply parameters magnitude, leads to similar induced bias. If we do not observe any benefits in top-1 accuracy, does best preserving the original network functions lead to better fairness in the predictions?

Setup The set of experiments conducted for this section extends over Hooker et al. (2021) work. In our setting, pruning is applied at the end of training and the pruned model is fine-tuned following the original training hyper-parameters with tuning on

the learning rate. More details on the training hyper-parameters can be found in Appendix C.2.

The model used is a ResNet-20 (He et al., 2015) trained on CIFAR10 and pruned at 95.4% . Note that pruning ratios were chosen accordingly to the Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2018) for the sake of comparison. Pruned models obtained following the LTH framework often constitute a strong baseline that many papers compare within the pruning literature. Experiments are run over 6 different seeds and pruning methods considered are MP and random with one iteration of pruning, and LM, GraSP_abs and Synflow with 100 iterations of pruning following a multi-stage approach (see Chapter 5 for more details on pruning methods and acronyms).

Evaluation The same metrics as in Hooker et al. (2021) are used to assess the fairness of the different pruned models. The impact of pruning is considered fair if the relative change in class accuracy between the dense and sparse models is uniform. Thus our null hypothesis can be defined as:

$$\mathcal{H}_0 : \frac{\beta^c_{\mathcal{D}}}{\beta^{\mathcal{M}}_{\mathcal{D}}} = \frac{\beta^c_{\mathcal{S}}}{\beta^{\mathcal{M}}_{\mathcal{S}}} \tag{6.2}$$

where $\mathcal{D}$ and $\mathcal{S}$ refer to the *dense* and *sparse* model, β^c to the per-class accuracy and $\beta^{\mathcal{M}}$ the overall accuracy. To verify the null hypothesis, a Welch's test is performed. The distribution of class accuracy shift is computed as

$$S^c = \{\beta^c - \beta^{\mathcal{M}}\} \tag{6.3}$$

where c is the class index, β^c the class accuracy and $\beta^{\mathcal{M}}$ the overall accuracy equivalent to the mean accuracy over all classes. For each class c, we perform a Welch's test between $S^c_{\mathcal{D}}$ – the dense model class shift – and $S^c_{\mathcal{S}}$ – the sparse model shift – to assess whether the mean between the two distributions diverges. If the *p-value* is <0.05, then there is a significant divergence between the dense and sparse models, which mean that the null hypothesis (Equation 6.2) is invalid and

the pruning criteria do not uniformly impact all the classes.

In addition to the mean divergence in class predictions, we compare the behaviour of different pruning methods investigated the Pruning Identified Exemplars (PIEs) generated. PIEs introduced by Hooker et al. (2021), measure the level of disagreement between the dense and sparse models. In other words, it is all the data points i for which the most frequent prediction y_i over a population of models, differ between the sparse and dense architecture:

$$PIE_i = \begin{cases} 1 & \text{if } y_{i,\mathcal{D}} \neq y_{i,\mathcal{S}} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

Those statistics, PIEs and Welsh's test, are computed over the test set for which there is an equal number of 1000 data samples for each class.

6.2.2 Experimental results

6.2.2.1 Pruning at the end of training

The behaviour of different unstructured pruning criteria is investigated under fairness metrics. Table 6.1 presents diverse statistics for different pruning methods when pruning is applied at the end of training and followed by a fine-tuning phase. For comparison, the performance of sparse model obtained under the Lottery Ticket Hypothesis (LTH) framework (see Chapter 4) known to be robust models are reported (Liebenwein et al., 2021).

When looking at the top-1 accuracy (second column) we can notice that most pruning criteria produce sparse networks that are in close range with performances oscillating around 80% for a Resnet20 pruned at 95.6%. LTH and MP are the best performing criteria with respectively 86.11% and 83.87%, while Random pruning is the worst performing criterion with 76.74%. When computing with a Welsh's test the number of classes whose prediction score shifted significantly between the baseline model and the sparse models (3rd column), we notice that apart from LTH model, a majority of classes were negatively impacted by pruning – 6-7 classes. By

having this cycle of pruning/fine-tuning, LTH seems to produce more robust models, but it could also be an effect of fine-tuning the pruned model from initialisation.

A Welsh's test assumes our class accuracy distribution follows a Gaussian distribution, however in our experimental setting due to the small number of experiments – 6 per pruning criteria – this assumption might not hold true. To confirm our initial observations, Figure 6.11 displays per class deviations $S^c = \{\beta^c - \beta^{\mathcal{M}}\}$ for different pruning methods, where β^c is the per-class accuracy and $\beta^{\mathcal{M}}$ the overall or mean accuracy. In an ideal fair and non-biased model, each class should have an equal prediction score resulting in little deviations from the overall (mean) accuracy illustrated by the *dotted line*. We can see that the baseline model (*blue squares*) is rather balanced as most classes observe only a small 5% accuracy deviation, except for the class *cat* for which the prediction is almost 10% off the mean accuracy.

If pruning criteria were to produce sparse models with similar outcomes as the initial dense network, then the shift in prediction should be equal to the one from the baseline resulting in all the points in figure 6.11 should overlap. If we can observe such tendency on the Frog class, for other classes (car, bird, cat, dog, ship) the impact of pruning is more pronounced, confirming that pruning is not applied uniformly across classes.

Lastly, we cannot conclude much from the number of PIEs – miss-classified

Resnet20 on CIFAR10 (end of training)			
Criterion	**Accuracy (%)**	**# Class shifted**	**# PIEs**
LTH	**86.11 ± 0.26**	3	1086
MP	83.87 ± 0.32	6	1218
Random	76.74 ± 1.68	6	1800
LM	78.51 ± 0.74	7	1762
$\|\|GraSP\|\|$	79.84 ± 0.69	7	1508
SynFlow	79.57 ± 0.83	6	1549

Table 6.1: Summary of different pruned model statistics. The 2nd column report the model accuracy after fine-tuning, the 3rd the number of class that shifted significantly from the baseline (dense) model, and finally the 4th column corresponds to the number of data points that observed a change in classifications between the dense and sparse networks outcome – note that CIFAR10 contains 10k datapoints. As a reference, the accuracy for the dense model is **90.90% ± 0.31**

examples – reported in Table 6.1. Because there is a significant drop in accuracy between the dense networks and the different sparse models, the number of PIEs is quite high. However, 247 data points are consistently miss-classified no matter the pruning criteria, Figure 6.15 present a snapshot of miss-classified examples. We can observe that these examples are either high contrasts images or unusual camera angles, and we could expect those examples to be harder to classify than others.

So far there seem to be little differences between pruning criteria when comparing them under fairness metrics. When comparing the divergence in class prediction between pruning criteria (Figure 6.12) we can observe two distinct groups: criteria preserving the original network function – LM, ‖GraSP‖ and SynFlow – that observe little divergence between one another with only 2 to 3 classes that resulted in a significant shift in prediction, and, *magnitude* heuristics –LTH and MP. Criteria preserving the original network functions produce models of similar quality with random pruning with only 0 to 1 classes that diverged significantly. This could either be an effect of more data points being required to regularise the observations, or some class or data point being much harder to classify and thus requiring more expressiveness from the network. By preserving the original network function we might remove useful noise to help classify harder data points.

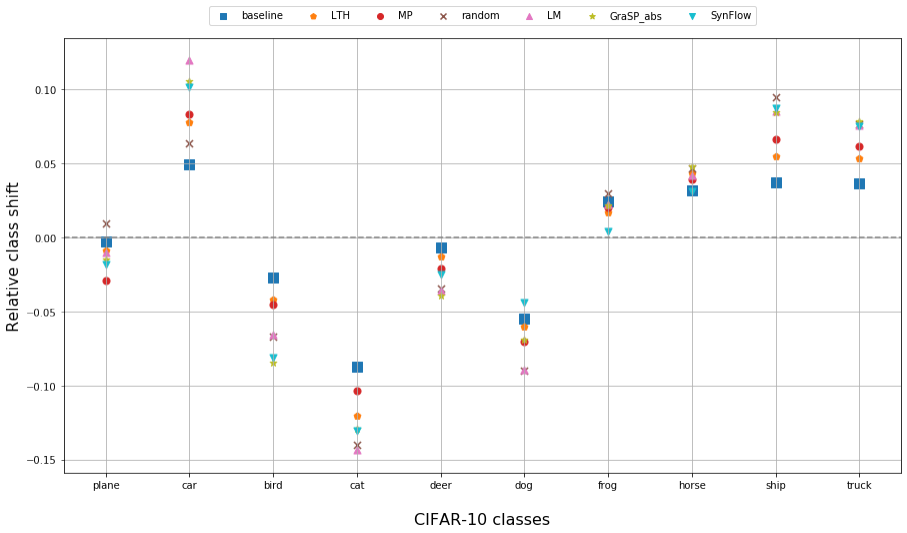


Figure 6.11: Per class prediction deviation for different pruning criteria when pruning is applied at the end of training. Classes are represented along the x-axis.

6.2.2.2 Pruning at initialisation

Because pruning was previously applied at the end of training, it is natural to suspect that biases induced by pruning might be reinforced by prior knowledge learned by the network. To test this hypothesis, pruning is applied at initialisation and we assess whether starting from a "blank" network helps preserving the original network outcome predictions.

Similar to Table 6.1, Table 6.2 presents a summary of the statistics investigated under various pruning criteria. Again the different sparse networks reach similar performances around 81% accuracy. For most of the pruning criteria, applying pruning at initialisation drastically reduces the number of classes that observe a significant shift in prediction accuracy. This number is reduced from 6-7 when pruning was applied after training to 3-4.

When investigating the per class deviation Figure 6.13, we do not observed any significant change between pruning at initialisation and end of training (Figure 6.11). Randomly pruning weights do not produce a uniform effect suggesting some examples are harder to classify than others. However, when comparing the shift in

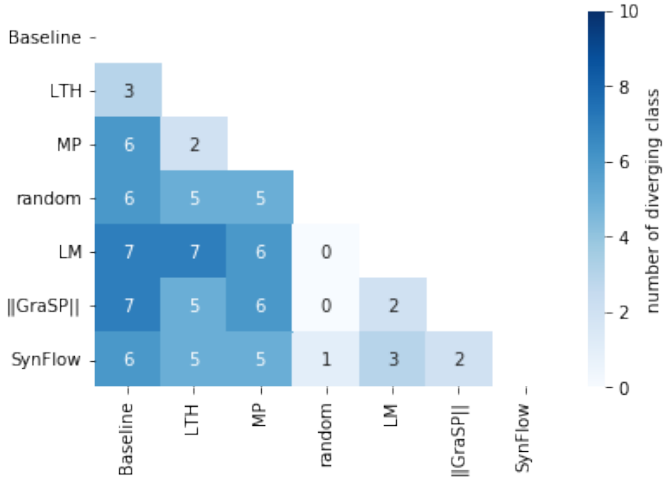


Figure 6.12: Number of classes with significant mean-shift class accuracy between different pruning criteria and the baseline – dense model, where 10 is the maximum number of classes. It can be observed that all the pruning criteria diverged significantly from the baseline indicating that pruning does not preserve the original model prediction distribution. Moreover, criteria preserving the original network function (LM, $\|GraSP\|$ and Synflow) observed similar behaviour with only a little shift in class prediction.

Resnet20 on CIFAR10 (initialisation)			
Criterion	**Accuracy (%)**	**# Class shifted**	**# PIEs**
LTH	**86.11 ± 0.26**	3	1086
MP	83.06 ± 0.26	5	1307
Random	79.44 ± 1.97	4	2340
LM	81.61 ± 0.43	6	1483
$\|\|GraSP\|\|$	80.36 ± 0.25	3	1445
SynFlow	81.54 ± 0.36	7	1398

Table 6.2: Summary of different pruned model statistics. The 2nd column report the model accuracy after fine-tuning, the 3rd the number of class that shifted significantly from the baseline (dense) model, and finally the number of data points that observed a change in classifications – note that CIFAR10 contains 10k datapoints. As a reference, the accuracy for the dense model is **90.90% ± 0.31**

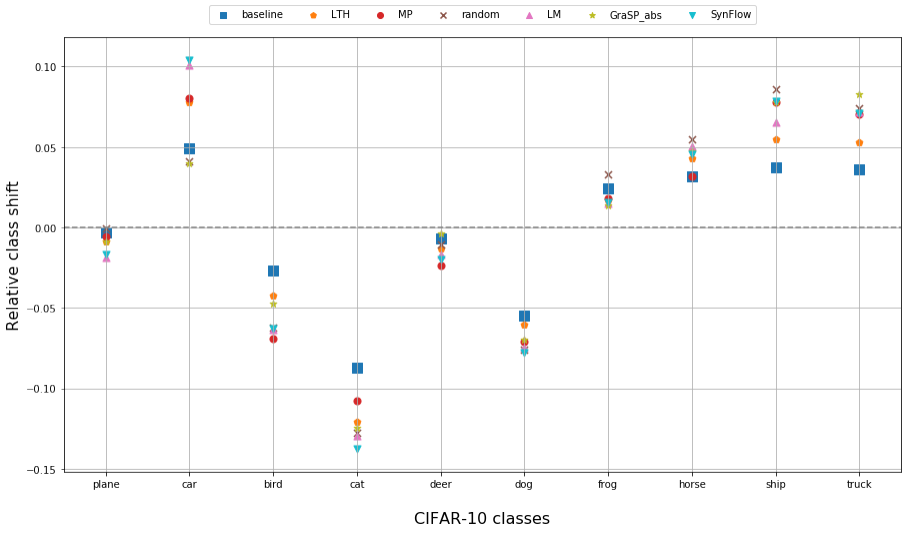


Figure 6.13: Similar to Figure 6.12 but for pruning at initialisation. Per class prediction deviation for different pruning criteria when pruning is applied before training.

prediction between pruning criteria (Figure 6.14), we notice that most models are aligned.

If pruning at initialisation produces slightly less biased sparse networks, we cannot clearly differentiate unstructured pruning criteria, they all produce sparse models of comparable quality.

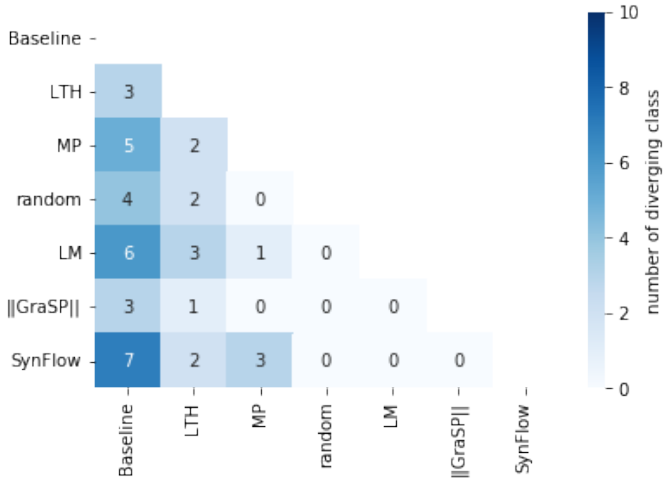


Figure 6.14: Comparison of number of classes with significant mean-shift class accuracy between pruning criteria. Similar to Figure 6.12 but for pruning at initialisation.

6.2.3 Discussion and limitations

Studying fairness and robustness in sparse models brings on a new perspective on the impact of removing parameters from the model beyond solely looking at the performance accuracy. If the accuracy can be preserved, sparse models usually reinforce biased present in the original dense model. It can become a tool to help assist practitioners in detecting potential failures of their model (Hooker et al., 2021, 2020).

Because magnitude heuristics – MP and LTH – remove the weakest connections, they are more prone to preserve the expressiveness of the neural network on difficult examples. On the other hand, criteria trying to preserve the original network function – LM, $\|GraSP\|$, SynFlow – are more sensitive to the data used to compute the importance measure. Real world data are noisy, the contribution of a data point cannot be estimated solely by its contribution to the loss (Hu et al., 2021).Real world data are noisy, the contribution of a data point cannot be estimated solely by its contribution to the loss (Hu et al., 2021). Those metrics are usually computed on the training set and all examples are considered "equally" when computing the criterion heuristic. But in practice not all examples are equal and some are harder to classify than others (see Figure 6.15). These criteria might be inclined to preserve weights related to popular examples leading to non-typical data points being

severely impacted by pruning: a cat in a weird position, an unusual camera angle picturing a horse from below, or high and low contrasts.

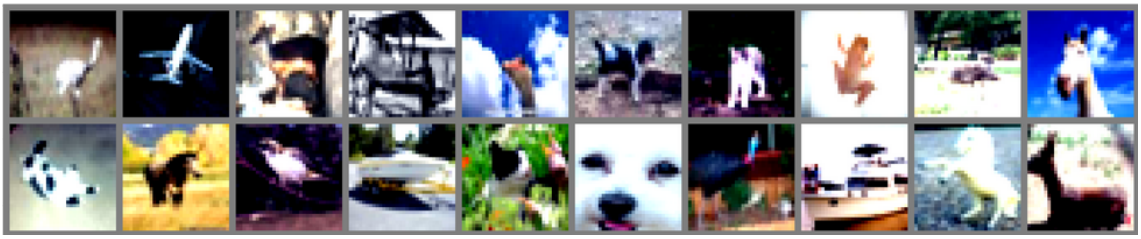


Figure 6.15: Example PIEs images obtained when pruning after training.

But the major factor influencing the fairness and robustness of sparse models is the pruning epoch. By pruning at initialisation, when the network has no prior knowledge, we can obtain more reliable pruned models. In particular for gradient-based criterion $\|GraSP\|$ that benefits greatly from pruning at initialisation. This can be explained by the fact that at initialisation because the neural network has not learned anything yet, the gradient for every data point has "equal" importance. At the end of training, some important parameters might have already reach their optimal value resulting in a zero-gradient. Also, because the gradient is trying to minimise the loss function, the optimisation might favour easy examples that produce stronger gradients and lead to a greater decrease for the cross-entropy score.

To alleviate this issue, a new pruning criterion could be designed trying to emphasize pruning weights related to *easy* data points and trying to preserve the weights related to *hard* examples. Data plays an important role in supervised learning approaches, researchers should be more careful with how they are used when computing importance measures to create more robust and fair sparse models.

6.2.3.1 Limitations

The results presented in this section are preliminary. They would benefit from a larger scale study with more pruning ratio explored. However, in our context, we were most interested in whether fairness metrics would constitute a better tool to differentiate unstructured pruning criteria and we demonstrated that it does not. This once again highlight the poor reliability of pruning criteria design.

6.3 Chapter Summary

This chapter presented exploratory works investigating the quality of pruning criteria beyond solely looking at the prediction accuracy. Two directions were explored: 1) the impact of parameterisation, to assess whether the lack of correlation between pruning objectives and pruned models performance was due to the high number of parameters or the complexity of modern architectures, and 2), the robustness of sparse model produced with different unstructured pruning criteria at preserving the original model prediction (fairness), to investigate if other characteristics could help differentiate between pruning criteria.

In Chapter 5 we demonstrated empirically that on modern architectures – VGG and PreActResNet – best preserving the original network loss or gradient flow did not correlate well with pruning performances. In other words, models good at preserving the original network functions were not necessarily reaching good accuracy after fine-tuning. In Section 6.1 we extended over this observation to explore whether the sole number of parameters or the complexity of modern architectures was causing poor correlation scores, studying how parameterisation impacted the correlation on wide (augmenting size of existing layers) versus deep (adding new layers) models on two toy models: Multi-Layers Perceptrons and Convolutional Networks. Our empirical study showed that the parameterisation, either deep or wide, did not impair the pruning performances much. Deeper networks were slightly more sensible than wider networks due to more non-linearity, but a clear drop in correlation performances could be observed when the pruned models were over-fitting during the fine-tuning phase. Because statistics computed to preserve the original network functions – loss $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ or gradient-flow $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ – are computed on the training set, the more the divergence between the validation and training sets, the lesser best preserving those properties will translate to better-performing networks. Thus pruning objectives reliant upon training data might not be a suitable candidate to create pruning criteria due to modern deep learning being in an over-fitting regime. Also this might suggests that pruning diminish the effect of regularisation

techniques.

One effective way to prevent over-fitting is to apply a high pruning ratio. In Section 6.1.3 we briefly reviewed the limitation of working with high pruning ratios presenting two common effects: drop in accuracy and layer collapsing. Another way to prevent over-fitting is to regularise the network but there is little research on regularisation methods for sparse training, especially in the context of pruning. This would constitute a promising direction for future research to improve existing pruning criteria.

Validation accuracy is not always the best tool to assess the performance of a pruned model. Recent research (Hooker et al., 2021) demonstrated that while preserving the original model accuracy, sparse models have a tendency to reinforce inherent biases from the training dataset, and increase weaknesses to adversarial attacks. In Section 6.2 we investigated whether fairness metrics would constitute a better tool to differentiate between different unstructured pruning criteria: magnitude-based, loss-preservation, or gradient-preservation. Maybe preserving the loss trajectory or gradient flow prevent diverging from the dense learning representations. We showed that in the context of pruning at the end of the training, no matter the criterion, pruning was not applied uniformly across classes and as a result, the outcome of the sparse model was of poorer quality than the original model, reinforcing biases (less fair).

However, when pruning was applied at initialisation, before any prior knowledge of the network, it seemed that preserving the gradient flow was helping reduce biases. We hypothesise that this was a result of every data point being considered equally as the network as not learning anything compare to pruning at the end of training where the network might have converged toward "easy" examples. In the end, all pruning criteria were producing sparse models of similar qualities and fairness can not be used to help identify the best unstructured pruning criterion

An alternate pruning criterion taking into account the disparities between training data points could become a powerful method to pruned efficiently deep neural

network. Weights related to popular and easy examples should be the ones to be pruned, while examples linked to complex data points should be preserved. Only then can we take full advantage of sparse neural networks.

In summary, current unstructured pruning criteria are not exploited to their full potential due to the over-fitting regime in which we train modern deep learning architecture. We need to rethink the way we design unstructured pruning criteria to take into account diverse difficulty of training data points and lack of regularisation during the fine-tuning phase, as well as take into consideration inherent biases in the training data.

Chapter 7

Conclusions

Deep Learning is a powerful tool to interpret and extract knowledge from a wide variety of data, from images to natural language. It has proven to be capable of reaching human-level performance, and has great potential applications in smart environments, smart cities, health and social welfare, and others. But neural networks are getting bigger, requiring ever increasing computational resources not only for training, but also for inference. This has significant implications for universal accessibility of AI technology with high costs, potential environmental impact through high power consumption and inability to use the models on mobile devices and low-power chips, demanding a constant update for existing hardware.

Unstructured pruning was developed as a tool to induce sparsity in a deep neural network to reduce the computational cost of running deep learning models. The goal is to remove parameters (i.e. set them to zero) based on some *importance measures* while maintaining good prediction accuracy, resulting in a high-performing network with a smaller computational footprint. Over the years, unstructured pruning has demonstrated to be a handy tool to help understand training dynamics in deep neural networks to design better and less demanding training frameworks.

Pruning has attracted lots of attention recently due to its simplicity and high potential to reduce the gap between academic research and AI deployment in production. This thesis investigated the state of unstructured sparsity for computer vision models, analysing the benefits and limitations of such an approach. In par-

ticular, a focus was put on the design of importance measures and pruning criteria to better understand what makes a good pruning criteria.

7.1 Hypothesis and Research Questions

7.1.1 RQ1: Pruning Framework

Pruning has originally been developed as a tool to reduce parameterisation in deep neural networks in order to produce more compute-efficient neural network architectures.The size and complexity of deep neural networks have been following an exponential growth since 2012 and the ImageNet breakthrough. This has greatly impacted the deployment of AI applications in real-world settings, where the compute is often limited and raised concerns about the sustainability of AI both socially and environmentally.

To alleviate this issue, many pruning approaches have been developed over the past decade. A wide variety of pruning frameworks have been proposed ranging from structured to unstructured pruning, pruning before, during, or after training, to retraining strategies. However despite the diversity of pruning methods, there is not a single method that consistently outperform the others. Finding the right pruning methodology depends on many factors: the level of compression we wish to achieve, the degree of accuracy we can afford to lose, the budget available to perform the pruning step, and the complexity of the task. Most methods require the pruned model to be fine-tuned, thus implying having the training data at hand. The complexity of the pruning heuristic, and the number of pruning cycles (training and fine-tuning) also need to be taken into consideration. For instance, the Lottery Ticket framework (Frankle & Carbin, 2018) achieves state-of-the-art compression and accuracy trade-off. However, to achieve approximately 95% size reduction by pruning 20% of the parameters each pruning step following an exponential decay, 15 cycles of training and fine-tuning are required, which has a significant impact for models that can take weeks to do one training iteration.

In summary, despite great progress pruning research is lacking a common benchmark to assess and compare the performance of all these compression approach. To help practitioners pick the best pruning method suited to their need, Blalock et al. (2020) proposed an open source PyTorch library to facilitate and standardised the evaluation of pruned models, but it is little used in the current literature.

Over years of research, pruning also shifted from application-oriented motives (lower computation costs, better generalisation, acceleration on low-power devices) to consideration towards empirical and theoretical understanding of deep neural networks (sparse training, dynamic training, role of over-parameterisation) thanks to unstructured pruning. When Frankle & Carbin (2018) and Liu et al. (2019) both demonstrated it was possible to train sparse neural networks from scratch to full accuracy, it attracted researchers to question what properties of those sparse neural networks lead to good training performance. Although many works explored and demonstrated the flexibility of sparsity at initialisation (Zhou et al., 2019; Paganini & Forde, 2020; Frankle et al., 2021), there is still a lot to learn. Unstructured pruning is a promising tool to understand what properties or parts of the network cause good performances, but more research is require in order to build more efficient pruning criteria and training framework.

7.1.2 RQ2: Integrity of unstructured pruning heuristics

The first step toward understanding how to create a good pruning framework is to evaluate the performance of existing pruning criteria. Evaluation of the pruned model performance often focuses on how far the model can be pruned without losing accuracy, but it does not assess the very foundation of pruning objectives and criteria. There is no clear gain from using one criterion in preference of another. However, a common problem to all pruning criteria is layer collapsing, when the architecture of the neural network is damaged beyond recovery by aggressively pruning a layer, preventing the pruned model from re-training properly. Pruning a small number of parameters at a time is a good approach to alleviate such a phenomenon.

A *pruning criterion* ranks parameters based on some *importance measure* derived from a specific *pruning objective*. In unstructured pruning there are three main pruning objectives: preserving the parameters' magnitude, preserving the dense network loss, or preserving the gradient flow from the original network. If such pruning objectives are efficient then best preserving the original network function should produce a better performing pruned model and the one objective leading to the greater accuracy improvement would constitute the most effective pruning approach.

In chapter 5 a comparison between different pruning criteria was conducted. To obtain different levels of loss or gradient-flow preservation within a same pruning criterion, we applied different tricks to enforce locality when approximating the original network function. We demonstrated that the previous statement was invalid and that best preserving the original network function (loss or gradient flow) does not lead to better performance after fine-tuning the pruned model, especially on modern architectures. This suggests that pruning criteria, and their pruning objective, are not adapted to work on current state-of-the-art architectures. *What causes pruning objective to fail at producing good sparse models on modern deep neural networks*?

Two main factors that differentiate modern architectures from original Multi-Layers Perceptions architectures are: the number of parameters (a hundred million against ten million), and the depth of the network inducing more non-linearity. Further analysis presented in Chapter 6 studied the effect parametrisation had on pruning objectives. We compared the behaviour of wide versus deep networks when augmenting the number of parameters adding neurons or filters on either existing layers or by adding new layers. We demonstrated that the overall number of parameters had little effect on pruning objectives. We were able to observe that better preserving the dense network functions led to better performing pruned models even on very wide architectures. Deeper networks were slightly more sensible than wider networks due to more non-linearity, but no clear drop in correlation performances could be observed. When the pruned models were over-fitting during the fine-tuning

phase, however, the correlation dropped drastically suggesting over-fitting causes pruning objectives to fail at producing efficient sparse models.

Because statistics computed to preserve the original network functions – loss $\Delta\mathcal{L}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})$ or gradient-flow $\Delta\|\mathbf{g}(\boldsymbol{\theta}, \Delta\boldsymbol{\theta})\|$ – are computed on the training set, the greater the divergence between the validation and training sets, the less benefit there is from best preserving those properties. By preserving the original network function too much, we remove diversity in the representations learned by the sparse network. They might be biased towards data points with high loss or gradient signal, with an impact on generalisation. To prevent such a phenomenon, more regularisation is required during the fine-tuning of sparse models. But in their current state, pruning criteria based upon loss or gradient preservation are not suitable candidates to create a reliable pruning model due to the over-fitting regime in which modern deep learning are trained.

7.1.3 RQ3: The impact of a sparse neural network on fairness

Despite their ability to achieve similar performance in terms of accuracy, dense and sparse models contain radically different numbers of parameters. It was initially believed that by reducing the amount of over-parameterisation, sparse models would memorise less and learn more generic and robust features. However, a recent study by Hooker et al. (2021) showed that it was quite the opposite. Sparse models obtained through unstructured magnitude pruning have a tendency to reinforce existing bias within the dataset, despite preserving a similar accuracy level.

Restricting the evaluation of a pruned model to a sole accuracy-sparsity trade-off limits our understanding of the pruned model's performance. Beyond solely looking at the best accuracy score after fine-tuning, assessing the fairness and robustness of the pruned model is crucial if they are to be deployed in real-world situations. Some natural suppositions when it comes to fairness in pruning could be that pruning criteria best preserving the gradient flow or loss landscape should be able to preserve

more generic features. While pruning at initialisation, when the network has not learned anything, should help preserve fairness.

In chapter 6, a study of different unstructured pruning criteria under the fairness umbrella was presented. If best preserving the original network function does not lead to better performance, do different pruning criteria produce different quality of pruned models? If a pruning criterion was to be fair, it should prune uniformly all classes. There should not be any shift in prediction between the dense and sparse network.

We demonstrated that similar to magnitude pruning presented in the original study by Hooker et al. (2021), other unstructured pruning criteria produce sparse model of lower quality. They do not prune classes uniformly and reinforce bias in the dataset. Criteria preserving the original network functions (loss or gradient flow) are the worst performing criteria, especially considering pruning at the end of training.

The pruning epoch has a major influence on the fairness of the pruned model. Model that are pruned later in training are more at risk of expanding biases in the existing data set, resulting in imbalanced per-class predictions. Even when pruning at initialisation, the criterion does not uniformly prune classes equally but it does improve fairness subsequently compared to pruning at the end of training because the network has not learned any prior representation of the data.

Criteria trying to preserve the original network function directly rely upon the data used to compute the importance measure. These criteria might be inclined to preserve weights related to popular examples, leading to non-typical data points being severely impacted by pruning. This raises concern about pruning methods that consider all data samples equally when some data are easier to learn than others. Thus, pruning might be inclined to favour classes related to easy data.

7.2 Contribution

The contributions of this thesis can be summarised as follow:

- A crowd-monitoring use-case study for the deployment of AI application in a real-world setting was presented in Chapter 3. Balancing processing between the edge – at the point of capture of data – and the cloud can help better preserve privacy and enhance responsiveness in the context of smart city applications, but deep learning models need to be compressed to be run on constrained devices where the compute power is limited.

- Chapter 4 presented an overview of the major works that shaped the field of pruning over the recent years. It highlighted the duality of pruning research being at the cross over application, where the aim is to lower the computational footprint, and theoretical research, where pruning is used to understand the limitation of deep learning training frameworks. If the same tools are employed, the motives are very different.

- An empirical investigation of the different unstructured pruning heuristics that can be found in the literature (presented in Chapter 5) demonstrated that unstructured pruning criteria are ill-defined and not adapted to large scale networks. Optimising to better preserve the original network functions – loss or gradient flow – (pruning objective) do not lead to better accuracy in the pruned model after fine-tuning, especially on modern neural network architectures.

- An experimental study evaluating the influence the network architecture has on the performance of the pruning criterion was presented in Chapter 6.1. Modern deep learning architectures are over-fitting even after most parameters have been removed (high ratios of pruning). Over-fitting during the fine-tuning phase of the pruned model causes the pruning objective to fail at consistently producing good sparse models. More regularisation for sparse training is required to allow pruning criteria computed on the training data set – loss or gradient flow approximation – to generalise better and produce robust sparse models after fine-tuning.

- Removing parameters from a deep neural networks can impair the quality of

the prediction. Beyond solely looking at the accuracy score after fine-tuning, in Chapter 6.2 we assessed through an empirical study how different unstructured pruning criteria influence the fairness of the pruned model. prediction. Two case study were investigated: (i) when the network has learn prior knowledge – pruning after training, and (ii) when the network has not learn prior knowledge – pruning before training.

When the dense network has learned prior knowledge, all pruning criteria where reinforcing existing bias in the dataset equally. To produce fair sparse models, pruning early in training and subsequently retraining the pruned model from initialisation or early on training, can help mitigate flaws induced by pruning.

7.3 Limitations

This thesis explored the foundation of pruning criteria for unstructured pruning in order to better understand how to build more efficient sparse networks. While pruning was initially designed to reduce the computational footprint of a deep neural network, it can also help uncover key structures at play during the training of deep neural networks.

The minimum number of parameters required in a deep neural network directly depends on the complexity of the task it is meant to solve. In this work, we mostly considered the case of small scale models and datasets. By limiting the scope of our study to smaller problems we were able to explore the inner mechanisms of pruning criteria without being limited by compute capacity. We demonstrated that the poor design of pruning criteria was a consequence of the over-fitting regime during the fine-tuning phase. In modern architectures and large-scale datasets, over-fitting is pro-eminent. Therefore, the results presented in this thesis are likely to remain valid in the context of larger-scale analysis.

A key element in the performance of the pruned model is the fine-tuning phase.

Right after pruning, the model suffers from a loss in accuracy that can be recovered shortly after re-training the sparse model. But fine-tuning, like training, is very sensitive to hyper-parameters. Hyper-parameters search can be expensive to run. If better accuracy can be obtained, this question the reliability of pruning criteria: Are good performance due to the distribution of sparsity (pruning mask)? However, one key hyper-parameter that need careful tuning is the learning rate. It allows the pruned model to be more flexible in relearning lost connections to solve the task.

It is the same for other pruning hyper-parameters such as the pruning ratio or the frequency of pruning. They are many factors that can influence the pruning performance but despite all, there are no pruning criteria that consistently outperform the others suggesting that all criteria are equal. Differences between pruning criteria can be observed under a specific framework, or when working with very high pruning ratios where certain criteria are more prone to layer collapsing. But it will not always be the same criterion performing the best, questioning the very foundation of pruning criteria and the way we evaluate pruned model performances.

7.4 The future of unstructured pruning

7.4.1 Practical Impact

Pruning has attracted a lot of attention over the past ten years with the promise of providing more efficient and compact deep neural networks. Its great flexibility and ease of use have made pruning a handy tool to explore the role and limits of parameterisation in deep neural networks. Unstructured pruning is not the best-suited approach when the goal is to compress the model to run on embedded devices. If it does greatly reduce the number of parameters, hence the required memory, unstructured pruning does not lead to a meaningful reduction in compute operations due to a lack of software support. There is still a lot of research to be done to improve software and hardware to support sparse training and inference, this research is essential to enable sustainable deployment of AI applications, lower training costs

and lesser carbon footprint. A great study summarising different approaches and software solutions to induce sparsity, structured and unstructured, can be found in Hoefler et al. (2021).

However, unstructured pruning has the potential to help understand training mechanisms to provide more resource-efficient deep neural networks. It demonstrated that it was possible to train a very sparse network from scratch and that parametrisation could be reduced prior to or early on training. There is a lot of flexibility within pruning, despite many pruning approaches and criteria developed over the years, none consistently out-perform the others. Is it the sparsity preserved per layer? The geometrical properties of the optimisation landscape? Too often pruning evaluation has been reduced to a pruning ratio/accuracy trade-off, omitting other factors such as fairness, robustness or consistency of the pruning mask. To create efficient pruning methods, we need to take a step back and understand what in the existing methods lead to good performance for sparse models. Because unstructured pruning drastically reduces the number of parameters, it is easier to compute network statistics.

This thesis demonstrated that pruning criteria for unstructured pruning are ill-defined and not adapted to modern deep learning architectures. The very nature of modern deep neural networks to be over-parameterised brings training in an over-fitting regime. However, most importance measures related to unstructured pruning criteria are designed based upon the training data, except for magnitude heuristic which is derived from the different steps taken during training. They either try to preserve the loss or gradient flow between the dense and sparse network. Thus measuring the importance of parameters under the training data, do not necessarily translate to better-performing pruned models because the training and validation set differ. This research shed light on the need for more regularisation during the fine-tuning of sparse models.

This also reminds us of the importance of the dataset. By trying to preserve the original network function, pruned models are more inclined to remove connections

essential for low-frequency or atypical data point that might represent noise in real-world data. Real world data are noisy, the contribution of a data point cannot be estimated solely by its contribution to the loss (Hu et al., 2021). To produce better and fairer pruned models, pruning criteria should take into account the complexity of the different data samples. Instead of considering all data point equally, pruning criteria should favour removing weights linked to easy or abundant data points, while preserving the weights for harder atypical examples. There is a lot of research to be done in that direction.

7.4.2 Research implications

Over years of research, pruning shifted from application-oriented motives (lower computation costs, better generalisation, acceleration on low-power devices), to consideration towards an empirical and theoretical understanding of deep neural networks (sparse training, dynamic training, role of over-parameterisation). This place unstructured pruning at the intersection of theoretical and applied research where goals and motives differ. For that reason, pruning is not well understood in a constantly growing community and is often seen as a mere *engineering tool* rather than a research domain on its own. Pruning has a great potential to help discern how to build more efficient neural network architectures, but to do so many sub-area of need to be explored from optimisation, regularisation, to hyper-parameters selection, masks flexibility, etc. In a highly competitive field where beating the state-of-the-art is often a key factor, the sparsity research community has struggled to gather around a common audience, venue or workshop.

In 2021, the first Sparsity in Neural Network workshop (SNN workshop[1]) was organised. After struggling to get the workshop accepted within one of the major conference venues, the organisers decided to organise the conference independently to provide a space for the community to exchange and discover the latest advancement in sparsity research. The event that took place online was very successful and a

[1]https://sites.google.com/view/sparsity-workshop-2021/

broad diversity of works could be advertised.

Because of its interdisciplinary nature – industry and academia – and duality – application-oriented and theoretical –, research in pruning is spread around. In addition to a dedicated workshop, a competition could help gather around latest research advancements and provide a benchmark needed for the comparison of new researches. This sparsity competition could set the foundation for evaluating compressed models assessing their reduction in floating-point operations, memory requirement, sparsity level and end accuracy, but also assessing robustness, fairness, and compute budget required for the pruning approach. It would be a great addition for the community to help practitioners navigate through the wide diversity of sparsity methods. Competitions are a great way to enhance and stimulate research as it has been proven with the success of the Imagenet at the initiative of many modern state-of-the-art architectures, and TrecVid competition for video captioning (Thornley et al., 2011; Awad et al., 2020).

7.4.3 Societal Impact

Unstructured Pruning is an area of tremendous growth and interest as deep neural networks continue to increase in ubiquity and ability. Pruning methods allow the deployment of neural networks requiring less computational resources and model storage. This may be beneficial in reducing the energy and environmental footprint required by a given system. They also enable the deployment of neural networks on embedded systems, facilitating a multitude of real-world applications, ranging from medical devices to weapon systems. Recent growth in this area has seen the emergence of pruning at the beginning of training. This could be highly beneficial for research equity as this would reduce the computational power required and the financial cost of training state-of-the-art neural networks for small academic and industry research labs. However, pruning at initialisation is still at the very early stages and current hardware and software infrastructures are lacking support for efficiently executing sparse neural networks.

An important social consequence for neural network models, especially as applied to the classification of personal data or computer vision, is the tendency towards bias and the perceived fairness of the systems often influenced by an imbalance in the training data. Pruned networks are prone to amplify potential bias and outliers in the data and should be monitored carefully (Hooker et al., 2020, 2021). On the other hand, they have potential use as a tool to evaluate the fairness of deep learning models

An important social consequence for neural network models, especially as applied to the classification of personal data or computer vision, is the tendency towards bias and the perceived fairness of the systems often influenced by an imbalance in the training data. The consequences of failure of the system using a pruned network are the same as the original network. However, pruning methods offer the opportunity to highlight potential bias, outliers or under-represented classes through analysis of the resulting network architecture and misclassifications. They, therefore, have potential use as a tool to evaluate the fairness of deep learning models (Hooker et al., 2020, 2021). In contrast, more research effort is needed to understand the implications of pruned models for increased risk of adversarial attack and model robustness for concept or class drift (especially in online learning situations). Our work highlights both the risk and the challenge of ensuring the optimum match of pruning criteria with a loss function that ultimately aims for greater integrity of unstructured pruning methods.

Appendix A

A.1 Smart Stadium experimental setups

Complementary information on experimental setup presented in Chapter 3 Section 3.1 and published in Ballas et al. (2018).

A.1.1 Hardware

The performance of each algorithm was tested on two different machines, a laptop and DELL Edge gateway 5000 with respectively an Intel `I5-3210M` (medium) and Intel `Atom E3825` (low) CPU. The main differences between the two CPUs are highlighted in Table A.1. We intentionally choose a i5 CPU to have a fair point of comparison with the Atom CPU. There is no doubt that an i7 or a Xeon CPU would have outperform the E3825 by far. The gateway runs with Ubuntu Core 16.04 OS.

	Model	**Release date**	**#core**	**#thread**	**CPU**	**cache**	**TDP**
Laptop	Intel Core i5-3210M	Q2 2012	2	4	2.50 GHz	3MB SmartCache	35W
Gateway	Intel Atom E3825	Q4 2013	2	2	1.33 GHz	1MB L2	6W

Table A.1: CPU specification for the Smart Stadium experiments.

A.1.2 Software

The software was implemented in Python using a multi-threaded approach to ingest raw images from the emulated IP camera, pre-process the frame and transfer the result to the Event Hub service in Azure as shown in figure A.1. The runtime was monitor with the `timeit` library. Docker was used to deploy the solution on the

gateway. The software was run multiple times on the laptop to ensure consistency before to be deploy on the gateway.

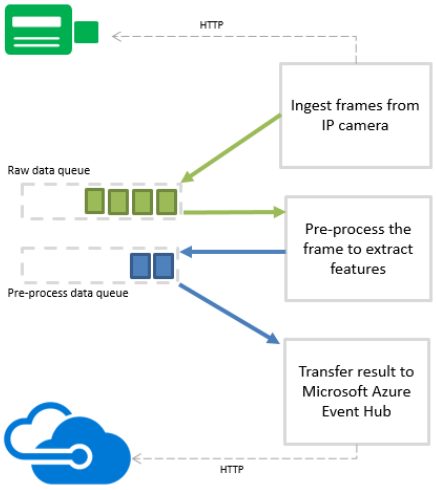


Figure A.1: Software architecture for crowd monitoring

A.2 Deep Learning Orchestration (DeepLO)

Complementary information on experimental setup presented in Chapter 3 Section 3.2.3.1.

A.2.1 DeepLo testbed hardware specifications

Table A.2 display the major hardware specifications for the testbed. Figure A.2 presents a list of the different metrics monitored during the experiments.

Metric	Formulation
CPU_CLOCKS	CPU_CLK_UNHALTED.THREAD
CPU_SLOTS	5 * CPU_CLOCKS
RECOVERY_CYCLES	INT_MISC.RECOVERY_CYCLES_ANY /2
MEMORY_BOUND	(CYCLE_ACTIVITY.STALLS_MEM_ANY + RESOURCE_STALLS.ANY) / CPU_CLOCKS
L1_DERIVED	(CYCLE_ACTIVITY.STALLS_MEM_ANY - CYCLE_ACTIVITY.STALLS_L1D_MISS) / CPU_CLOCKS
L2_DERIVED	(CYCLE_ACTIVITY.STALLS_L1D_MISS - CYCLE_ACTIVITY.STALLS_L2_MISS) / CPU_CLOCKS
HIT_FRAC	MEM_LOAD_RETIRED.L3_HIT / (MEM_LOAD_RETIRED.L3_HIT + 7* MEM_LOAD_RETIRED.L3_MISS)
L3_DERIVED	(1 - HIT_FRAC) * CYCLE_ACTIVITY.STALLS_L2_MISS / CPU_CLOCKS
MISSPRED	(UOPS_ISSUED.ANY - UOPS_RETIRED.RETIRE_SLOTS + 5 * INT_MISC.RECOVERY_CYCLES) / CPU_SLOTS
SPECULATION	
EXTERNAL_MEMORY_BOUND	CYCLE_ACTIVITY.STALLS_MEM_ANY
RETIRING	UOPS_RETIRED.RETIRE_SLOTS / CPU_SLOTS

Figure A.2: List of the different hardware metrics monitored to assess the cost of running deep learning workloads on constrained devices.

	Testbed Hardware			
	frp1	up2	i5nuc	i7nuc
Hardware	Intel Xeon Processor E3-1275 v5	Intel Pentium Processor N4200	Intel Core™ i5-7260U Processor	Intel Core™ i7-7567U
Processor	E3-1275 v5	N4200	i5-7260U	i7-7567U
Year	Q4'2015	Q3'2016	Q1'2017	Q1'2017
# of cores	4	4	2	2
# of threads	8	4	2	2
Base Frequency	3.6 GHz	1.10 GHz	2.2 GHz	3.4 GHz
Cache	8 MB SmartCache	2 MB L2	4 MB	4 MB
Memory in settings	31.3 GB	7.6 GB	15.6 GB	15.6 GB

Table A.2: Specifications of the different hardware in the DeepLO testbed. Note that frp2 is a more recent version of the frp1 aquired later in the project.

Appendix B

B.1 Details on the Experimental Setup (section 5.4

)

B.1.1 Setups

Datasets We use the MNIST dataset (LeCun et al., 1998), and hold-out 10000 examples randomly sampled from the training set for validation. We also use CIFAR10 (Krizhevsky et al., 2009), where the last 5000 examples of the training set are used for validation, and we apply standard data augmentation (random cropping and flipping, as in He et al. (2016b)) during training phases. For ImageNet (Deng et al., 2009), we follow the experimental setting of Goyal et al. (2017).

Network Architectures On MNIST, we use a MLP of dimensions 784-300-100-10, with Tanh activation functions. On CIFAR10, we use both: a VGG11 (Simonyan & Zisserman, 2015), equipped with ReLUs (Nair & Hinton, 2010), but no Batch Normalisation (Ioffe & Szegedy, 2015); and the PreActResNet18, which is the 18-layer pre-activation variant of residual networks (He et al., 2016b). MLP leverages Glorot & Bengio (2010) as initialization while the the weights of VGG11 and PreActResNet18 are initialized following He et al. (2015), and the biases are initialized to 0. On ImageNet (Deng et al., 2009), we use a ResNet-50 (He et al., 2016a) with Batch Normalization, and follow the initialization strategy described in (Goyal et al., 2017).

B.1.2 Experiments

For the MNIST and CIFAR10 experiments, the network is first trained for a fixed number of epochs, using early stopping on the validation set to select the best performing network.The hyper-parameters used for training are selected via grid search (before even considering pruning). Then we prune a large fraction of the parameters.

For OBD, LM, and QM we randomly select, at each iteration of pruning, 1000 examples (10 mini-batches) from the training set to compute the gradients and second order terms of the models.[1] For SynFlow we also used 1000 examples to compute SynFlow saliencies following code provided by Tanaka et al. (2020). For GraSP we used the Grasp dataloader from Wang et al. (2020) code to select which parameters to discard.

Finally, we retrain the network using exactly the same hyper-parameters as for the initial training.

For ImageNet, we uses the exact same hyper-parameters than Goyal et al. (2017).

MLP on MNIST We train the network for 400 epochs, using SGD with learning rate of 0.01, momentum factor of 0.9, l2 regularisation of 0.0005 and a mini-batch size of 100. We prune 98.85% of the parameters.

VGG11 on CIFAR10 We train the network for 300 epoch, using SGD with a learning rate of 0.01, momentum factor of 0.9, a l2 regularisation of 0.0005 and a mini-batch size of 100. The learning rate is divided by 10 every 60 epochs. We prune 95.6% of the parameters.

PreActResNet18 on CIFAR10 We train the network for 200 epochs, using SGD with a learning rate of 0.1, momentum factor of 0.9, a l2 regularisation of 0.0005 and a mini-batch size of 100. The learning rate is divided by 10 every 70

[1] Using 1000 examples or the whole training set made no difference in our experiments. Using less examples started to degrade the performances, which concord with the observations of Lee et al. (2019)

epochs. We prune 95.6% of the parameters.

ResNet50 on ImageNet For ImageNet, we train a ResNet50 using 8 V100 GPUs. The total mini-batch size is 256, and we train our baseline network for 90 epochs. The learning rate schedule is identical to Goyal et al. (2017): a linear warm-up in the first 5 epochs and decay by a factor of 10 at epochs 30, 60 and 80. We then prune 70% of the parameters. After pruning, we fine-tune the models for 90 epochs using a learning rate of $1e^{-3}$. For LM, QM, OBD, we investigates the following hyper-parameter values: $\pi \in \{1, 100\}$, $\lambda \in \{1e^{-3}, 1e^{-1}, 0, 10, \}$. 1600 examples are used to compute the first and second order terms of the linear and quadratic models.

B.1.3 Pruning Framework

We apply pruning at the end of training. Pruning is applied globally, we use a pruning ratio of `pr=98.85%` for MLP and `pr=95.6%` for VGG11 and PreActResnet18 on CIFAR10 to align with pruning ratio presented in the lottery ticket framework (Frankle & Carbin, 2018). Each pruning phase is followed by a fine-tuning step. Note that because of their convergence assumption, OBD and OBS advocate for fine-tuning after each stage of pruning. Since LM and QM, or gradient-based criteria GraSP and SynFlow, are not based on this assumption, they should perform well in this proposed framework. See algorithm 1 for more details about the pruning framework.

Appendix C

C.1 Supplementary materials for Wide and Deep networks experiments (Section 6.1)

C.1.1 Training Hyper-parameters

MLP on MNIST We train the network for 100 epochs, using SGD with a learning rate of 0.01, momentum factor of 0.9, l2 regularisation of 0.0005 and a mini-batch size of 100. We prune 98.85% of the parameters.

Convnets on CIFAR-10 We train the network for 200 epochs, using SGD with a learning rate of 0.01, momentum factor of 0.9, an l2 regularisation of 0.0005 and a mini-batch size of 100. The learning rate is divided by 10 every 80 epochs. We prune 95.6% of the parameters.

Convnets on MNIST We train the network for 160 epochs, using SGD with a learning rate of 0.01, momentum factor of 0.9, an l2 regularisation of 0.0005 and a mini-batch size of 100. The learning rate is divided by 10 every 60 epochs. We prune 95.6% and 98.85% of the parameters.

α	Performance accuracy (%) for different pruning criteria (width scaling)						
	ø	**MP-1**	**LM-1**	**LM-100**	**QM-100**	**GraSP-100**	**SynFlow-100**
1	1.53 ± 0.04	2.86 ± 0.1	6.22 ± 1.2	2.94 ± 0.2	2.87 ± 0.15	3.82 ± 0.18	2.85 ± 0.12
2	1.51 ± 0.03	3.02 ± 0.2	12.42 ± 5.9	2.55 ± 0.2	2.6 ± 0.09	3.74 ± 0.19	2.85 ± 0.12
3	1.5 ± 0.05	3.19 ± 0.2	7.91 ± 1.5	2.44 ± 0.2	2.48 ± 0.16	3.49 ± 0.11	2.45 ± 0.05
4	1.51 ± 0.04	2.99 ± 0.2	7.77 ± 3.4	2.25 ± 0.1	2.31 ± 0.03	3.64 ± 0.25	2.55 ± 0.13

Table C.1: Summary of the best validation error across different *width scaling* for different pruning criteria. The number following each criterion name indicates the number of local iteration (π) performed during pruning. We observe that all networks are reaching good performances indicating no problems encountered during pruning or fine-tuning.

α	Performance accuracy (%) for different pruning criteria (depth scaling)						
	ø	**MP-1**	**LM-1**	**LM-100**	**QM-100**	**GraSP-100**	**SynFlow-100**
1	1.53 ± 0.04	2.86 ± 0.1	6.22 ± 1.2	2.88 ± 0.2	2.87 ± 0.15	4.08 ± 0.03	2.85 ± 0.12
2	1.54 ± 0.1	3.63 ± 0.8	88.71 ± 0.27	2.98 ± 0.13	3.04 ± 0.15	6.36 ± 4.42	2.65 ± 0.12
3	1.55 ± 0.07	60.02 ± 40.55	88.71 ± 0.27	3.17 ± 0.18	3.38 ± 0.09	9.62 ± 3.97	2.93 ± 0.19
4	1.48 ± 0.047	88.71 ± 0.27	88.71 ± 0.27	3.23 ± 0.1	3.22 ± 0.17	15.30 ± 4.26	2.94 ± 0.2

Table C.2: Summary of the best validation error across different *width scaling* for different pruning criteria. The number following each criterion name indicates the number of local iteration (π) performed during pruning.

C.1.2 Models accuracy

Tables C.1 and Table C.2 reference the different accuracy obtained after fine-tuning for the MLP networks trained on MNIST. We can observe that some networks (MP-1 and LM-1) failed to retrain properly for deep architectures.

C.1.3 Fine-tuning

C.1.3.1 Depth Fine-tuning

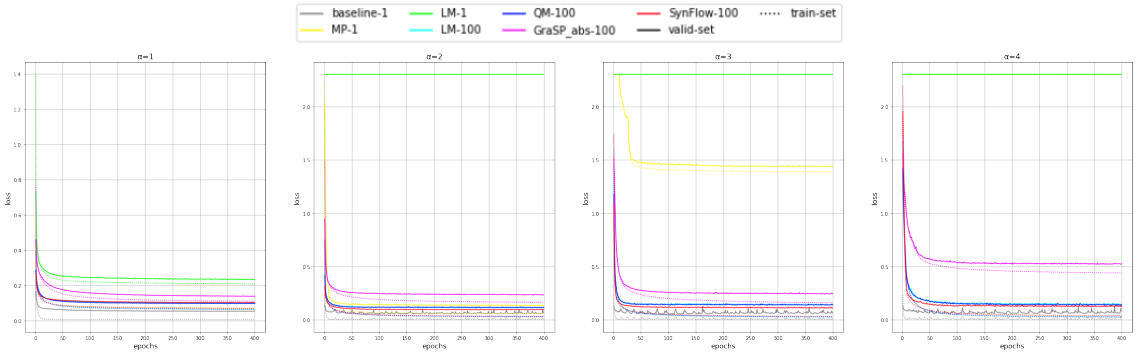
Similar to Figure 6.7 in Chapter 6, Figure C.1 displays the training and validation *loss* for fine-tuning curves for deep models scaled by adding new layers.

C.1.3.2 Width Fine-tuning

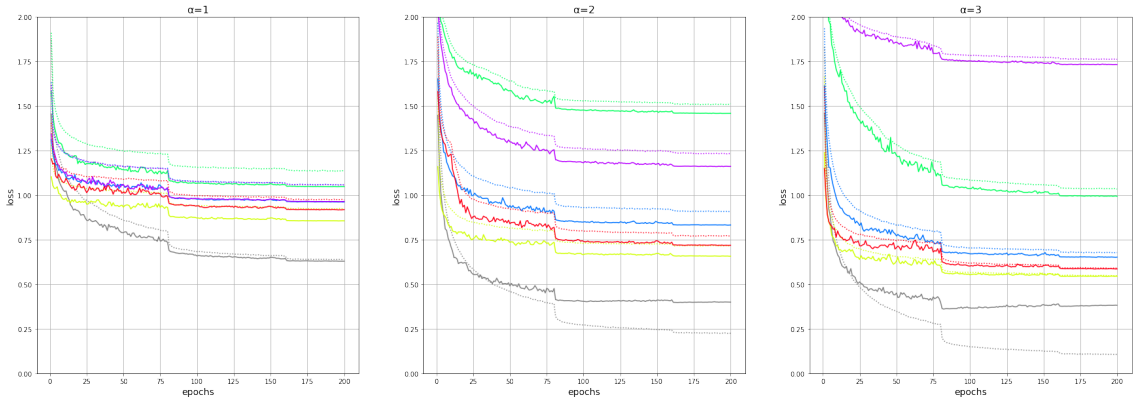
Similar to Figure 6.7 in Chapter 6, Figure C.2 displays the training and validation *errors* fine-tuning curves for wide models scaled on the width – adding new units on existing layers. Likewise, Figure C.3 displays the training and validation *loss* fine-tuning curves for model scaled on the width.

C.1.4 Layer collapsing

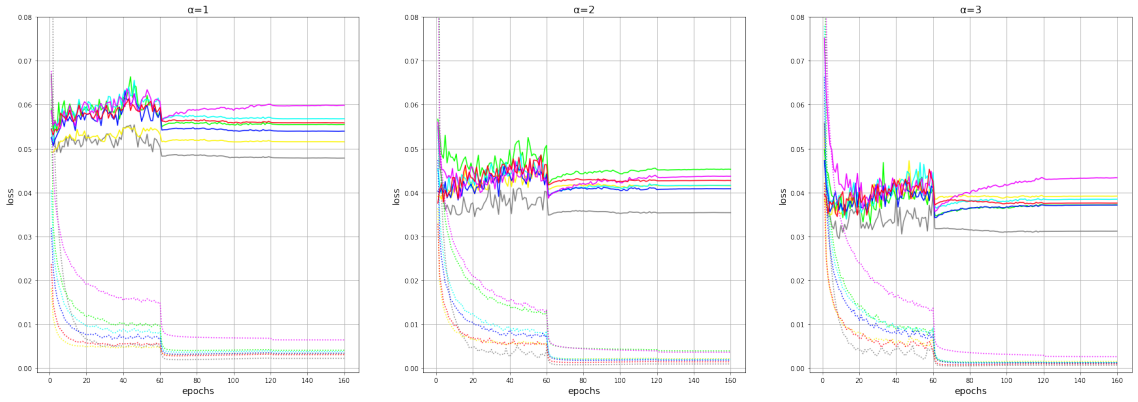
Figure C.4 display the percentage of remaining parameters per layer for a VGG16 network pruned following different pruning ratios (colour). Each column corresponds to a different pruning criterion, while each line corresponds to a different pruning epoch. We can observe that layer collapsing do not only happen at initialisation and is even more at risk of happening for smaller pruning ratios when pruning is applied later during training.



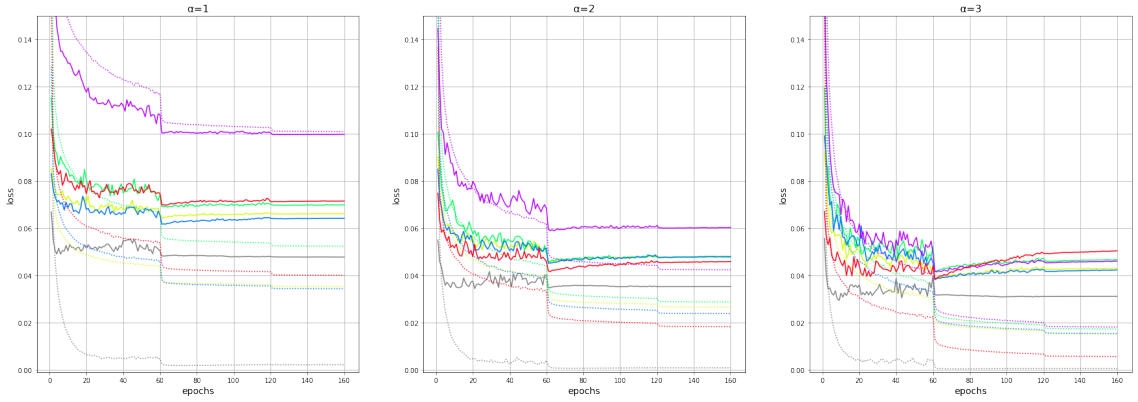
(a) MLPs on MNIST



(b) Convnets on CIFAR-10

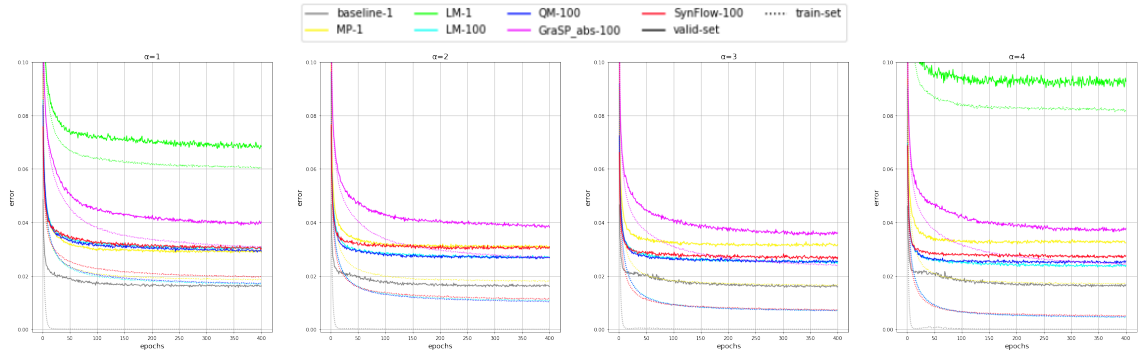


(c) Convnets on MNIST with pr=95.6%

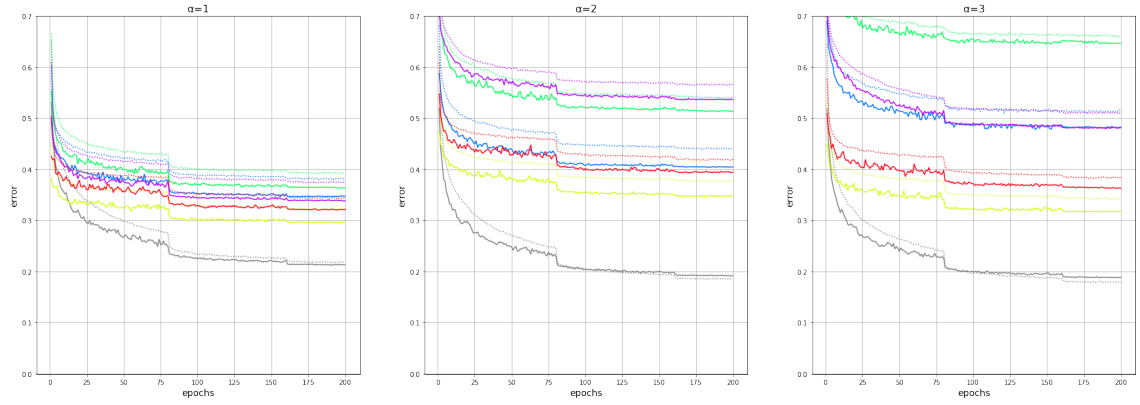


(d) Convnets on MNIST with pr=98.85%

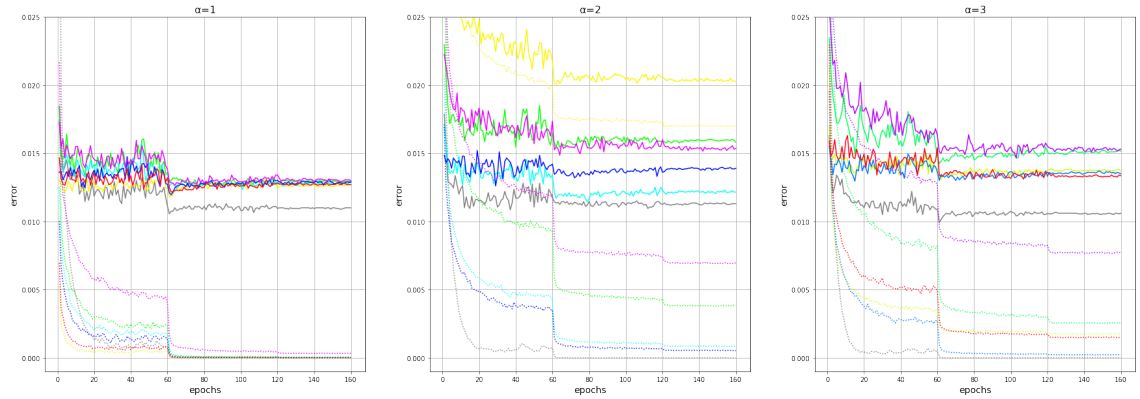
Figure C.1: Fine-tuning curves (loss) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance.



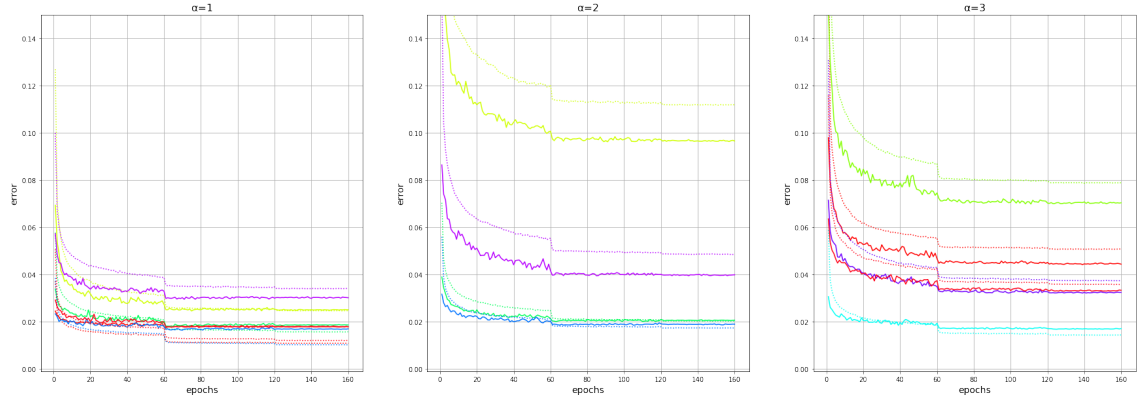
(a) MLPs on MNIST



(b) Convnets on CIFAR-10

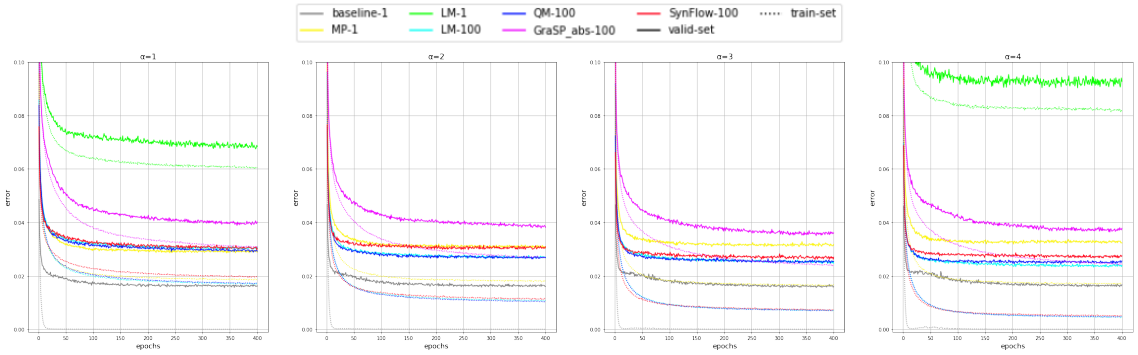


(c) Convnets on MNIST with pr=95.6%

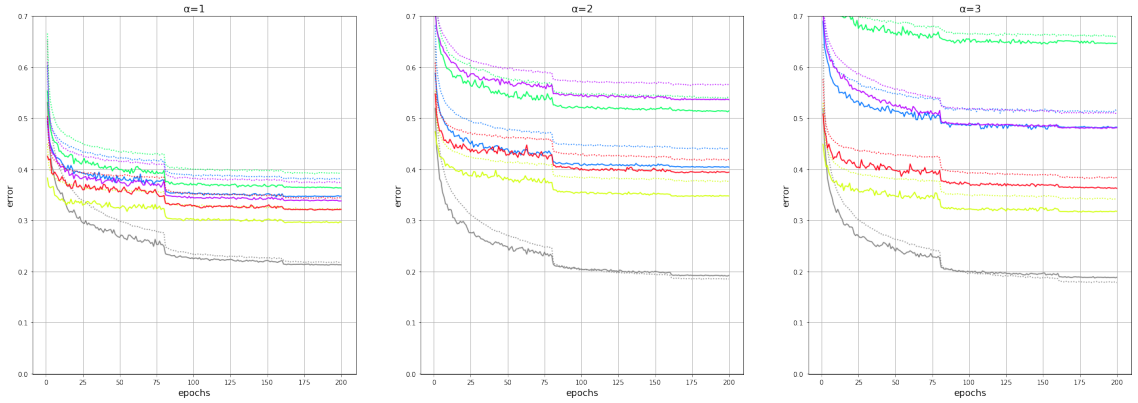


(d) Convnets on MNIST with pr=98.85%

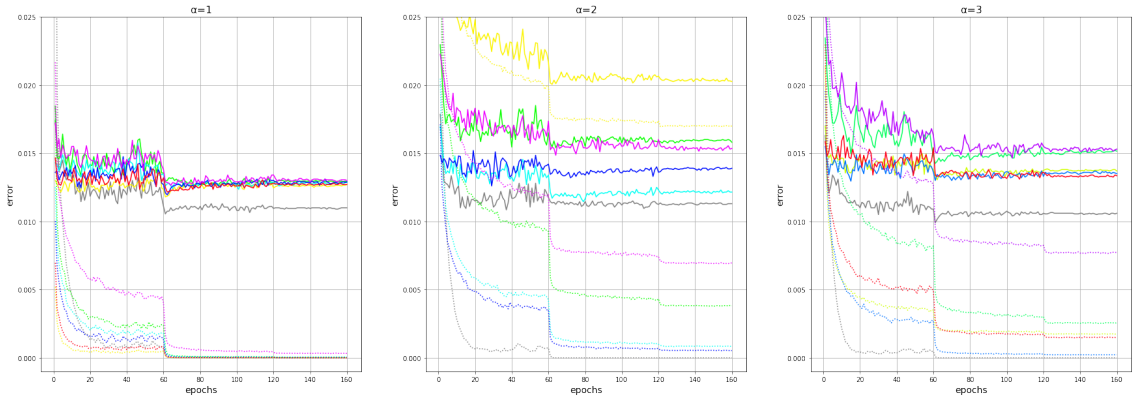
Figure C.2: Fine-tuning curves (classification error) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance.



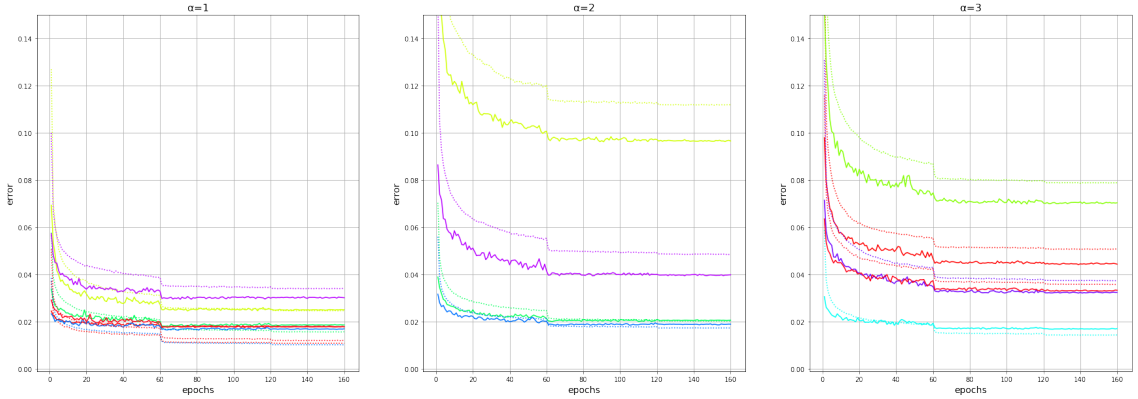
(a) MLPs on MNIST



(b) Convnets on CIFAR-10



(c) Convnets on MNIST with pr=95.6%



(d) Convnets on MNIST with pr=98.85%

Figure C.3: Fine-tuning curves (loss) for different model architectures scale on the width. Each curves is average over 5 seeds, for readability we do not display the variance.

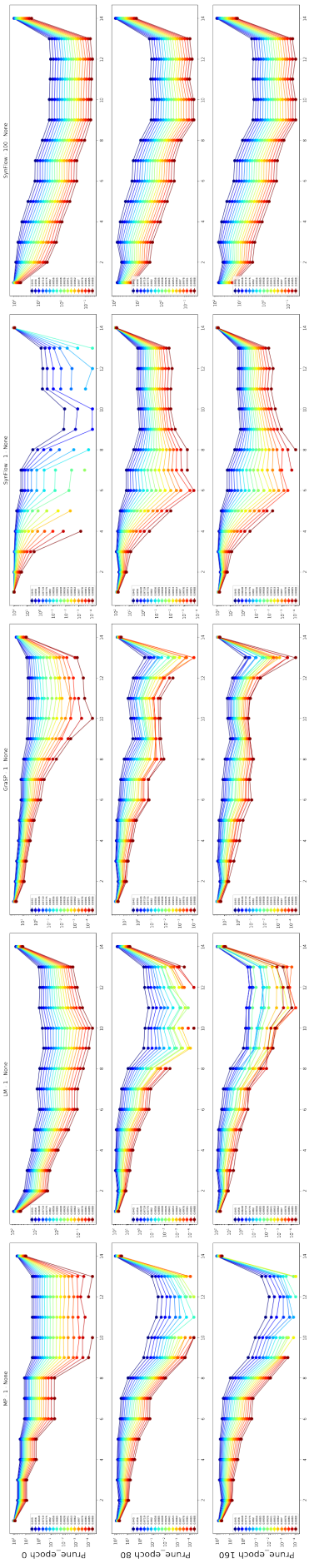


Figure C.4: Ratio of parameters remaining per layer (x axis) for a VGG19 architecture when pruned before training (top), at the middle of training (middle) or after training (bottom). Magnitude pruning is applied in this example. Different colors correspond to different ratio of pruning, with blue being a small pruning ratio and red high pruning ratio.

C.2 Supplementary materials for fairness in pruned networks (section 6.2)

C.2.1 Training Hyper-parameters

ResNet20 on CIFAR-10 We train the network for 160 epochs, using SGD with a learning rate of 0.1, momentum factor of 0.9, an l2 regularisation of 0.0005 and a mini-batch size of 100. The learning rate is divided by 10 every 60 epochs. We prune 95.4% of the parameters.

Bibliography

REGULATION (EU) 2016/ 679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL - of 27 April 2016 - on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/ 46/ EC (General Data Protection Regulation).

Kale ab Tessera, Sara Hooker, and Benjamin Rosman. Keep the gradients flowing: Using gradient flow to study sparse network optimization, *arXiv preprint*, 2102.01670, 2021.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`, 2015.

Alessandro Achille, Giovanni Paolini, and Stefano Soatto. Where is the information in a deep neural network?, *arXiv preprint*, 1905.12213, 2020.

Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dy-

namics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020.

Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frederic Petrot. Ternary neural networks for resource-efficient AI applications. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2017-May, pp. 2547–2554. IEEE, 5 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966166.

Jose M Alvarez and Mathieu Salzmann. Compression-aware Training of Deep Networks. *NIPS*, 2017.

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute. URL `https://openai.com/blog/ai-and-compute/`, 2018.

Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization, *arXiv preprint*, 1812.03981, 2018.

George Awad, Asad A Butt, Keith Curtis, Yooyoung Lee, Jonathan Fiscus, Afzal Godil, Andrew Delgado, Jesse Zhang, Eliot Godard, Lukas Diduch, et al. Trecvid 2019: An evaluation campaign to benchmark video activity detection, video captioning and matching, and video search & retrieval, *arXiv preprint*, 2009.09984, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, *arXiv preprint*, 1607.06450, 2016.

David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question?, *arXiv preprint*, 1702.08591, 2018.

Camille Ballas, Mark Marsden, Dian Zhang, Noel E. O'Connor, and Suzanne Little. Performance of video processing at the edge for crowd-monitoring applications. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 482–487, 2018.

Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3: 517–532, 2021.

Brian Bartoldson, Ari Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. *Advances in Neural Information Processing Systems*, 33:20852–20864, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off, *arXiv preprint*, 1812.11118, 2019.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pp. 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL `https://doi.org/10.1145/3442188.3445922`.

Yoshua Bengio. Learning Deep Architectures for AI. *Machine Learning*, 2(1):1–127, 2009. doi: 10.1561/2200000006.

Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Binary-densenet: Developing an architecture for binary neural networks. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 1951–1960, 2019.

Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6: 64270–64277, 2018.

Johan Bjorck, Carla P. Gomes, and Bart Selman. Understanding batch normalization. *CoRR*, abs/1806.02375, 2018.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big data and internet of things: A roadmap for smart environments*, pp. 169–186. Springer, 2014.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware, *arXiv preprint*, 1812.00332, 2019.

William Chan, Mitchell Stern, Jamie Ryan Kiros, and Jakob Uszkoreit. An empirical study of generation order for machine translation, 1910.13437.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16306–16316, 2021.

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott

Yang. Adanet: Adaptive structural learning of artificial neural networks. In *International conference on machine learning*, pp. 874–883. PMLR, 2017.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications, *arXiv preprint*, 1412.7024, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in neural information processing systems*, 26, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint*, 1810.04805, 2018.

Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, and Luca Fanucci. An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick. *Int. J. Reconfigurable Comput.*, 2019:7218758:1–7218758:13, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020a.

Utku Evci, Yani A. Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win, *arXiv preprint*, 2010.03533, 2020b.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *CoRR*, abs/1803.03635, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *CoRR*, abs/1912.05671, 2019.

Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training, *arXiv preprint*, 2002.10365, 2020.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark?, *arXiv preprint*, 2009.08576, 2021.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, *arXiv preprint*, 1902.09574, 2019.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, 31, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

Anna Golubeva, Behnam Neyshabur, and Guy Gur-Ari. Are wider nets better given the same number of parameters? *CoRR*, abs/2010.14495, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, *arXiv preprint*, 1706.02677, 2017.

Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37 (3):362–386, 2020.

Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 9 2013.

Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, *arXiv*, 1510.00149, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*. 2015b.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

Zhezhi He, Boqing Gong, and Deliang Fan. Optimize deep convolutional neural network with ternarized weights and high accuracy. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 913–921. IEEE, 2019.

Yashar D Hezaveh, Laurence Perreault Levasseur, and Philip J Marshall. Fast automated analysis of strong gravitational lenses with convolutional neural networks. *Nature*, 548(7669):555–557, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, *arXiv preprint*, 1503.02531, 2015.

Sepp Hochreiter, Yoshua, Fakultit F/jr Informatik, Yoshua Bengio, Paolo Frasconi, and Jfirgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 2003.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. Characterising bias in compressed models, *arXiv preprint*, 2010.03058, 2020.

Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget?, *arXiv preprint*, 1911.05248, 2021.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017.

F. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton paperbacks. Princeton University Press, 2002. ISBN 9780691090658.

Niel Teng Hu, Xinyu Hu, Rosanne Liu, Sara Hooker, and Jason Yosinski. When does loss-based prioritization fail?, *arXiv preprint*, 2107.07741, 2021.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Stanislaw Jastrzebski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of dnn loss and the sgd step length, *arXiv preprint*, 1807.05031, 2019.

Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. The break-even point on optimization trajectories of deep neural networks, *arXiv preprint*, 2002.09572, 2020.

Rupp Karl. CPU, GPU and MIC Hardware Characteristics over Time, *Blog post*, https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/, 2013.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pp. 491–507. Springer, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning, *arXiv preprint*, 1910.09700, 2019.

Jennifer Langston. From conversation to code: Microsoft introduces its first product features powered by gpt-3, https://blogs.microsoft.com/ai/from-conversation-to-code-microsoft-introduces-its-first-product-features-powered-by-gpt-3/, May 2021.

César Laurent, Camille Ballas, Thomas George, Nicolas Ballas, and Pascal Vincent. Revisiting loss modelling for unstructured pruning, *arXiv preprint*, 2006.12279, 2020.

Duong H. Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants, *arXiv preprint*, 2105.03193, 2021.

Yann LeCun. Who is afraid of convex optimization? NIPS - Workshop on Efficient Learning. URL `https://cs.nyu.edu/ yann/talks/lecun-20071207-nonconvex.pdf`, 2007.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the Institute of Radio Engineers*, 86(11):2278–2323, 1998.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, *arXiv preprint*, 1810.02340, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, *arXiv preprint*, 1608.08710, 2017.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5958–5968. PMLR, 2020.

Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems*, 3:93–138, 2021.

Suzanne Little, Dian Zhang, Camille Ballas, Noel E O'Connor, David Prendergast, Keith Nolan, Brian Quinn, Niall Moran, Mike Myers, Clare Dillon, et al. Understanding packet loss for sound monitoring in a smart stadium iot testbed. In *Proceedings of the First ACM International Workshop on the Engineering of Reliable, Robust, and Secure Embedded Wireless Sensing Systems*, pp. 40–45, 2017.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV 2017*, 8 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning, *arXiv preprint*, 1810.05270, 2019.

Ada Lovelace. Notes upon L. F. Menabrea's "Sketch of the Analytical Engine invented by Charles Babbage", 1842.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Ekdeep Singh Lubana and Robert P. Dick. A gradient flow framework for analyzing network pruning, *arXiv preprint*, 2009.11839, 2021.

Aapo Markkanen. Iot analytics today and in 2020. *Competitive Edge from Edge Intelligence. ABI Research, Oyster Bay, NY*, 2015.

Mark Marsden, Kevin McGuinness, Suzanne Little, and Noel E. O'Connor. Fully Convolutional Crowd Counting On Highly Congested Scenes, 1612.00220, 2016.

Mark Marsden, Kevin McGuinness, Suzanne Little, and Noel E O'Connor. Resnetcrowd: A residual deep learning architecture for crowd counting, violent behaviour detection and crowd density level classification. In *2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS)*, pp. 1–7. IEEE, 2017.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better, *arXiv preprint*, 2106.08962, 2021.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pp. 2498–2507. PMLR, 2017.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *Advances in neural information processing systems*, 32, 2019.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ2). In *Doklady an ussr*, volume 269, pp. 543–547, 1983.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. *Advances in neural information processing systems*, 31, 2018.

Michela Paganini and Jessica Zosa Forde. Bespoke vs. prêt-à-porter lottery tickets: Exploiting mask similarity for trainable sub-network finding, *arXiv preprint*, 2007.04091, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34, 2021.

Bryan A. Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. Neural parameter allocation search, *arXiv preprint*, 2006.10598, 2021.

David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, Cambridge, UK, 2 edition, 2017.

J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Stephan Rasp, Michael S. Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning, *arXiv preprint*, 2003.02389, 2020.

Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey of machine learning accelerators. *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2020.

Eva Mohedano Robles. Phd thesis: Deep image representations for instance search. *SIGMultimedia Rec.*, 10(2), August 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 2002.

Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, *arXiv preprint*, 1409.1556, 2015.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

Nitish Srivastava, Geoffrey E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path mobile automl: Efficient

convnet design and nas hyperparameter optimization. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):609–622, 2020.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, 2020.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning, *arXiv preprint*, 2007.05558, 2020.

Clare Thornley, Shane Mcloughlin, Andrea Johnson, and Alan Smeaton. A bibliometric study of video retrieval evaluation benchmarking (trecvid): A methodological analysis. *J. Information Science*, 37:577–593, 12 2011.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, *arXiv preprint*, 1607.08022, 2017.

United Nations. Department of Economic and Social Affairs. Population Division. *The World's Cities in 2016 : Data Booklet.* United Nations, Department of Economic and Social Affairs, Population Division, [New York NY?], 2016.

Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NeurIPS 2011*, 2011.

P C Veena, Paulsy Tharakan, Hima Haridas, K Ramya, Riya Joju, and T S Jyothis. Smart street light system based on image processing. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–5. IEEE, 3 2016.

Shikhar Verma, Yuichi Kawamoto, Zubair Fadlullah, Hiroki Nishiyama, and Nei Kato. A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues. *IEEE Communications Surveys & Tutorials*, (c):1–1, 2017.

Bob D de Vos, Floris F Berendsen, Max A Viergever, Marius Staring, and Ivana Išgum. End-to-end unsupervised deformable image registration with a convolutional neural network. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pp. 204–212. Springer, 2017.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pp. 1058–1066. PMLR, 2013.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pp. 6566–6575. PMLR, 2019.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow, *arXiv preprint*, 2002.07376, 2020.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22–31, 2021.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated

residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S. Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp, *arXiv preprint*, 1906.02768, 2020.

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 558–567, 2021.

Wenyuan Zeng, Yuwen Xiong, and Raquel Urtasun. Network automatic pruning: Start nap and take a nap, *arXiv preprint*, 2101.06608, 2021.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 365–382, 2018.

Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 589–597. IEEE, 6 2016.

Qibin Zhao, Masashi Sugiyama, Longhao Yuan, and Andrzej Cichocki. Learning efficient tensor representations with ring-structured networks. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 8608–8612. IEEE, 2019a.

Ruizhe Zhao, Brian Vogel, and Tanvir Ahmed. Adaptive loss scaling

for mixed precision training. *CoRR*, abs/1910.12385, 2019b. URL `http://arxiv.org/abs/1910.12385`.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, *arXiv preprint*, 1710.01878, 2017.

List of publications

- Suzanne Little, Dian Zhang, **Camille Ballas**, Noel E. Oćonnor, David Prendergast, Keith Nolan, Brian Quinn, Niall Moran, Mike Myers, Clare Dillon, and Tomas Meehan. **Understanding packet loss for sound monitoring in a smart stadium IoT testbed.** In *Proceedings of the First ACM International Workshop on the Engineering of Reliable, Robust, and Secure Embedded Wireless Sensing Systems* (FAILSAFE'17). Association for Computing Machinery, New York, NY, USA, pp. 40–45.

- **Camille Ballas**, Mark Marsden, Dian Zhang, Noel E. O'Connor, and Suzanne Little. **Performance of video processing at the edge for crowd-monitoring applications.** In *2018 IEEE 4th World Forum on Internet of Things* (WF-IoT), 2018, pp. 482-487

- César Laurent, **Camille Ballas**, Thomas George, Nicolas Ballas, and Pascal Vincent. **Revisiting loss modelling for unstructured pruning.** In *arXiv preprint*, and presented under *"Investigating Loss-modelling pruning criteria for unstructured pruning"* at the ICLR 2021 - SEDL Workshop, 2020.

- **Camille Ballas**, César Laurent, Thomas George, Nicolas Ballas, Suzanne Little and Pascal Vincent. **Assessing Criteria Integrity for Unstructured Pruning of Deep Neural Networks.** In *preprint*, 2021