POLITECNICO DI TORINO

Master's Degree in DATA SCIENCE AND ENGINEERING

Master's Degree Thesis

Structured Pruning for Efficient Convolutional Networks

Supervisors

Prof. GIUSEPPE BRUNO AVERTA

Prof.ssa BARBARA CAPUTO

Ph.D. FABIO CERMELLI

Dott.ssa CLAUDIA CUTTANO

Candidate

GIOVANNI CALLERIS

OCTOBER 2023

Abstract

In recent years, the size of Deep Learning models has increased. This trend has been allowed by the advancements in this research area and the development of the hardware targeted for its use. It has led to better results overall. Bigger models, having more parameters, can adapt better to the given task than their smaller counterparts. However, small models are helpful in many real-life scenarios, if not mandatory. They are needed for energy consumption constraints, speed requirements, and hardware limitations.

The lottery ticket hypothesis states that a trained model may contain a sub-network whose performance would be similar to the one of the whole model. In this work, we try to find this sub-network through pruning, a technique for removing parameters, starting from a pre-trained model. This work proposes a method for applying structured pruning on Deep Learning models in Computer Vision settings, specifically in image classification. The intent is to keep the original model's performance unchanged while reducing the number of parameters, and thus the model size, in a structured way: it does not remove single parameters but whole channels, avoiding making the network sparse and hard to optimise on hardware. Usually, pruning methods base their decision upon a chosen criterion; the technique introduced in this work is instead trained to select which parameters to prune and which to keep, thus not introducing a bias in the selection. This is done by the OutputAnalyser class that wraps the model, substitutes the 2D convolutional layers with a PruningConv2D module, and, at the end of the pruning phase, reinserts the now pruned convolutional layers. Each PruningConv2D contains one of the original convolutional layers and a model, which we will call the pruning-model. At training time, the pruning-models' predictions will select which of the original layer parameters to keep and discard. We use their last prediction at test time to decide which channels to keep. Moreover, we avoid using other external and hard-to-balance losses to do this.

The accuracy score obtained on the CIFAR-10 dataset is 92.84 %: removing 33.23 % of the parameters and reducing the FLOPs by 38.23 %, the error increased to 7.16 %, starting from 6.47 % of the original ResNet-110 model on this dataset. We used a specific set of pruning-models for this method. Still, this approach opens up new possibilities since it allows the user to change their structure: they can be deepened and enlarged with no additional computational cost for the final pruned model. This methodology could also be extended to different kinds of layers and other Deep Learning tasks.

Acknowledgements

I want to thank Professor Giuseppe Averta for the continuous support and kind attention he has given me, Professor Barbara Caputo for the enthusiasm for this subject she has been able to convey during her course, Fabio Cermelli and Claudia Cuttano for guiding me through the hurdles we encountered during the realisation of this Thesis. I also want to thank the Vandal research group for welcoming me during these months.

A special thanks goes to my family for always being there during quiet periods and through stormy waters. I want to toast to the troubles I met for helping me recognise what true friendship means. To those friends, I am really grateful.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep Learning is a fascinating field. We can find its roots in mathematical modelling and informatics. It attempts to induce intelligent behaviour into machines by imitating how natural neural networks work. In recent years, it has gained more and more relevance concerning other Artificial Intelligence algorithms and the general public, thanks to its successes.

1.1 Thesis Objective

This Thesis aims to reduce the size of a Deep Learning model for the image classification task.

Many use cases need and involve the use of edge devices to have the processing of the data closer to the data source. There may be many reasons to do this: reducing privacy and security issues and avoiding the need for an internet connection, jointly cutting down power consumption. However, the model must match the hardware constraints. These devices are smartphones, IoT devices or, more generally, those with stricter resource constraints than a typical Deep-Learning-ready setup. The one we focused on in this Thesis belongs to the family of pruning algorithms. Specifically, we introduce a new method for doing structured pruning through an all-differentiable approach, thus letting us use Deep Learning from top to bottom.

1.2 An Introduction To Deep Learning

An Artificial Neural Network is composed of neurons, the basic unit of the network. A neuron is composed of parameters and connections. The connections determine its position in the network concerning the others and make computation possible: they connect it to its previous and following neurons. The parameters are usually

multiplied by the inputs. Adding all these elements reduces the resulting output vector to a scalar number. The result is then passed to the following neuron.

Neurons are grouped in layers. Each layer extracts features from its inputs and passes them to the following. The features extracted are, at first, very simple and local. Still, later in the network, one layer after another, they can become more abstract and complex and capture the general meaning of what is being analysed, whether a picture or a text.

Layers are connected: more layers make a network deeper; more neurons inside layers make a network wider. Usually, features are normalised between layers to keep the extracted values in a normal range and passed through an activation function to induce non-linearity in the network. The deeper and wider the model, the more it can fit the given task, as it can adapt better to the function it has to mimic.

It is trained on data to learn this function: during the learning phase, it aims to reduce the error between its output and the ground truth.

Much data representing the target function is required to fit a model properly. Otherwise, it could over-fit the task by adapting too much to the examples seen, thus losing generality, by exploiting its capacity to learn.

In mathematics, the gradient of a function in a given point is directed toward the steepest increase of that function. By taking the opposite direction, we can move down toward a minimum. In Deep Learning, we want to minimise the error. Thus, we can change the parameters according to the negative value of the gradient.

The updates of the parameters are scaled down by a value called "learning rate" and other parameters according to the optimisation function chosen. This is necessary because it is impossible to analyse the whole data together, but it has to be split into batches: a subset of the training data. Each time the network processes a batch, we update the parameters. When the whole training data has been analysed once, an epoch has been done. The training process is composed of many epochs.

The learning process needs smooth updates: the gradient must have neither big nor small values. Otherwise, it would incur the so-called exploding or vanishing gradient problems.

Also, the network's architecture can be modified to avoid these problems, especially when going deeper. The most famous example of such a change in architecture is the residual connection introduced by He et al. [1]. In this architecture, layers are grouped into blocks. The input of each block is added to the output of the whole block. The residual connection thus helps the gradient flow through the shortcut path, called "skipped connections". It also helps the network itself: the relevant features extracted by the previous block will pass to the following one, and those that are not will be put to zero and, likely, be overridden by the features extracted inside the block or vice versa.

Thanks to this and other techniques, networks have thus become wider and deeper in recent years, making it possible to achieve better results and tackle new challenges. More extensive networks were also made possible by the advances in the hardware that specifically support these kinds of operations, like Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), making this kind of calculation feasible in a reasonable amount of time.

1.3 Deep Learning On Edge Devices

Deep Learning on edge devices, like smartphones and sensors, means moving the processing of the data from remote servers to where data is generated or acquired. Getting the processing closer to the data source helps reduce latency, improve privacy and increase security, eliminate the need for an internet connection, and decrease power consumption. The price we must pay to obtain these benefits is subject to device constraints. There are different ways to fulfil these requirements: knowledge distillation, pruning, and quantisation are some of those.

1.3.1 Knowledge Distillation

Knowledge Distillation is a technique that reduces the model's size, requiring two models: a pre-trained teacher and a student.

The teacher and student model usually have similar architecture, but the teacher is wider and deeper. The main idea is to teach the student to make predictions as similar to the ones of the teacher as possible. To this end, many techniques have been proposed, some of which require a specific loss that reduces the difference between the distribution of the predictions: The Kullback-Leibler divergence loss [2].

1.3.2 Pruning

Pruning algorithms aim to reduce the parameters in a pre-trained model while preserving the original model's metric results, like classification accuracy.

There are different ways to implement it. The first distinction must be stated between structured and unstructured pruning. The former aims to remove entire neurons from the layers. The latter removes single parameters from each neuron. The main difference resides in the fact that structured pruning while modifying the cardinality of the feature dimension of the output of each layer keeps its representation dense: some features extracted by the previous layer may be ignored or, after an optimisation phase, wholly removed, but both input, parameters and output are still dense tensors. On the other hand, unstructured pruning removes single parameters. Thus, the model's parameters will become sparse. This

sparsity implies that if not provided with specific hardware optimised for this kind of computation, the pruning operation will have no practical benefits as many accelerators will cast the sparse tensor representation to a dense one.

1.3.3 Quantisation

Quantisation is another technique to reduce the size of the pre-trained model. The main idea is to compress numbers from 32 to 16 bits, 8 or less. Cutting down the number of bits used will reduce the size of the parameters on the memory to half, a quarter, or less of the original 32-bit model's size. It is crucial to remember that even in this case, specific hardware is required. If the hardware does not support any particular optimisation for these data types, they will be automatically converted back to floating point of 32 bits. On the other hand, if the hardware supports it, the inference speed may drastically increase. Unless we only need to save memory on the device, this approach requires quantising all the tensors that pass through the model: inputs, outputs and the ones created at intermediate steps.

1.4 Computer Vision

Computer Vision is a predominant field in machine learning that empowers image and video analysis. It is a very active research branch with applications often used in the industry. The most famous example is its use to enable autonomous driving vehicles. Many tasks can be done in this field. Some of these are:

- Image Classification, whose aim is to classify the element caught in a picture;
- Object Detection, which consists of identifying and locating objects in a picture through bounding boxes;
- Semantic Segmentation, where the objective is to classify each element in a picture, doing it pixel by pixel.

We have used our pruning algorithm for the classification task. The layer pruned in this work is the PyTorch [3] implementation of the Conv2d [4]. This layer is based on the sliding window mechanism: a kernel, or filter, of parameters slides on a picture, analysing a set of pixels at a time. The result is a new picture made by "pixels" with channels with varying cardinality. The cardinality of the channel dimension depends on the number of kernels in the Convolutional layer. For example, we can have three input channels at the first convolution in a network if we analyse RGB (Red, Green and Blue) images, various output channels and a five-by-five kernel. This setting will lead to a tensor-parameter

with shape: $(3, \text{output channels}, 5, 5)$. It is crucial to focus on the size of this tensor-parameter. We could get many values in this tensor using even a small number of output channels. Most of all, later in the network, we usually have a more significant number of input channels, which may quickly increase the total number of parameters in the network. For example, with 16 input and output channels and a 3×3 kernel, we are getting 2304 values. This, generally, would not even be considered a wide layer.

Chapter 2

Literature Overview

Pruning, in neural networks, is a technique that aims to reduce the model's size. Its roots rely on the lottery ticket hypothesis [5], which states that a model has inside a sub-network whose performance would be similar to the one of the original. Models might be redundant by construction, for example, when using dropout [6]. Dropout is a regularisation technique that randomly sets some feature values to zero to increase the network's robustness and reduce over-fitting. Networks may also be over-parameterised, meaning there are more parameters and neurons than the number of features useful to extract to fit the task. Pruning may improve the model generalisation since fewer parameters will make it harder for the network to over-fit the task. As mentioned in the introduction, pruning can be structured and unstructured. The main difference resides in the granularity at which pruning acts: at the level of the parameters if unstructured, at the filter, channel, or layer level if structured. The common trait between all these techniques is their reliance on a criterion.

Since this thesis focuses on structured pruning, the following briefly discusses some of these algorithms. Following the structure of a recent survey [7] on this topic, the methods that are more relevant for this work are summarised according to this survey's schema:

2.1 Weight-Dependent;

2.2 Activation-Based;

2.3 Regularization;

2.4 Dynamic Pruning.

This work combines ideas from both the Activation-Based 2.2 and Dynamic Pruning 2.4 methods.

2.1 Weight-Dependent

Criterion depending on parameters, also referred to as weights, do not depend on the input data. These, generally, are less computationally expensive. There are two main kinds: *filter norm* and *filter correlation* algorithms.

2.1.1 Filter Norm

In filter norm algorithms, the l_1-norm or the l_2-norm is computed on the layer's filter values. Filters with low-norm values are pruned since they are supposed to contribute less to the network's final output.

Pruning Filters for Efficient ConvNets

In Pruning Filters For Efficient ConvNets (PFEC) [8], the l_1-norm is applied.
Here is the formula of the l_1-norm:

$$\|x\|_1 = |x_1| + |x_2| + ... + |x_n| \tag{2.1}$$

Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks

In Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks (SFP) [9], the l_2-norm is applied.
Here is the formula of the l_2-norm:

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{2.2}$$

2.1.2 Filter Correlation

Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration

Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration (FPGM) [10] specifically aims to prune redundant ones, not those that contribute less. The geometric median is used to obtain this information.
The formula of the geometric mean is the following:

$$G = \sqrt[n]{x_1 x_2 ... x_n} \tag{2.3}$$

COP: Customized Deep Model Compression via Regularized Correlation-based Filter-Level Pruning

COP: Customized Deep Model Compression via Regularized Correlation-based Filter-Level Pruning (COP) [11] analyses cross-layer filter importance using the

Pearson correlation coefficient. To normalise the layer's values, max-normalization is used to scale. This paper calculates this coefficient between layers to remove redundant filters. The less relevant ones are pruned after ranking the filters upon the computed metric.
The formula of the Pearson correlation coefficient [12] is the following:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.4}$$

Here, x and y are two variables; their signed counterparts are their means.

2.2 Activation-Based

Activation-based, or activation channel pruning, removes filters based on the activation maps: the result of the convolutional layer. They can be divided into three classes:

2.2.1 The Current Layer;
2.2.2 The Adjacent Layer;
2.2.3 All Layers.

2.2.1 The Current Layer

Channel Pruning for Accelerating Very Deep Neural Networks

In Channel Pruning for Accelerating Very Deep Neural Networks [13], pruning is reduced to a minimisation problem:

$$\arg_{(\beta,\ W)} \min \frac{1}{2N}\left\|Y - \sum_{i=1}^{c}\beta_i X_i W_i^{\top}\right\|_F^2 + \lambda||\beta||_1 \tag{2.5}$$

$$\text{subject to } ||\beta||_0 \leq c',\ \forall i||W_i||_F = 1 \tag{2.6}$$

Where $||...||_F$ is the Frobenius norm:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{2.7}$$

Here X is the input, Y the layer's output, and W the weights. λ is a coefficient positively bound to the number of zeros, and β is the coefficient vector for channel selection composed of scalar masks.

HRank: Filter pruning using high-rank feature map

HRank: Filter pruning using high-rank feature map [14] proposes an approach where the Singular Value Decomposition (SVD) of each layer's output is first computed. The SVD formula is the following:

$$A = U\Sigma V^\top, \tag{2.8}$$

where U is a matrix containing the orthonormal eigenvectors of $AA^\top$, V is a matrix containing the orthonormal eigenvectors of $A^\top A$, and Σ is the diagonal matrix containing the root of the non-negative eigenvalues of $AA^\top$. This operation is done on a sub-sample of the dataset. Then, the filters are ranked by their importance. Finally, only the top-k filters are kept in the network.

2.2.2 The Adjacent Layer

ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression

ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression [15], analyses the following layer's activation map to prune the current layers. This is done with a greedy algorithm that minimises the reconstruction error and satisfies the sparsity needed.

Runtime Neural Pruning

Runtime Neural Pruning (RNP) [16] dynamically prunes, at runtime, the filters in the layers. This is not to use the same weights for simple and hard-to-infer images. By analysing the complexity of the previous layer feature map through global pooling followed by a shared Recurrent Neural Network [17], they exploit a reward function for pruning filters, together with the original loss, to make the model prune filters.

2.2.3 All Layers

NISP: Pruning Networks using Neuron Importance Score Propagation

In NISP: Pruning Networks using Neuron Importance Score Propagation [18] proposed a metric based on the Final Response Layer (FRL). This determines the importance of the neurons preceding the last layer. These relevance scores are computed by propagating through the network weights initialised to one, starting from the last layer, based on the NISP algorithm. Once this metric is obtained, neurons with low importance are pruned based on a ranking algorithm.

2.3 Regularization

2.3.1 Regularization on BatchNorm Parameters

Learning efficient convolutional networks through network slimming

Learning efficient convolutional networks through network slimming [19] introduced a scaling factor γ for each channel. These parameters are multiplied by the output of the corresponding channel. Then, the network and these parameters are trained while sparsity regularisation is imposed on these γs. Pruning is done on those channels whose corresponding scaling factors have a low value. Then, the network is fine-tuned.
The minimisation problem becomes the following:

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma) \tag{2.9}$$

Here x is the input, y the target, and W the weights. While the first part of the equation is the standard loss function, the second part is the sparsity-induced penalty, balanced by the parameter λ.

2.3.2 Regularization on Extra Parameter

Data-driven sparse structure selection for deep neural networks

Data-driven sparse structure selection for deep neural networks [20] introduced the scaling factors θ, which are gates: if the value is below a certain threshold, it will be set to 0, else 1. The reported formula is the following:

$$g(\theta) = \begin{cases} 0 & \text{if } \theta < \text{Threshold} \\ 1 & \text{otherwise} \end{cases} \tag{2.10}$$

These parameters are applied to all the inner structures of the network. To optimise this function, they also introduced the Accelerated Proximal Gradient optimiser.

2.3.3 Regularization on Filters

Learning structured sparsity in deep neural networks

Learning structured sparsity in deep neural networks [21], based on the fact that removing channels implies the removal of the corresponding filters and following layer's inputs, used two separate regularisation terms for filter-wise and channel-wise pruning. The formulas are the following:

$$\sum_{n_l=1}^{N_l} ||W^{(l)}_{n_l,:,:,:}|| \tag{2.11}$$

$$\sum_{c_l=1}^{C_l} ||W^{(l)}_{:,c_l,:,:}|| \tag{2.12}$$

here $W \in \mathbb{R}^{N_{l+1} \times N_l \times K_l \times K_l}$

2.4 Dynamic Pruning

2.4.1 Filter-level dynamic

Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks

Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks (SFP) [22] introduced the idea that instead of directly removing the channels, or filters, from the network, we can reduce them to zero, allowing them to be modified in subsequent iterations. The masks are generated based on the l_2-norm.

Feature Map Analysis-Based Dynamic CNN Pruning and the Acceleration on FPGAs

The following article was not taken from the survey [7].

In Feature Map Analysis-Based Dynamic CNN Pruning and the Acceleration on FPGAs (FMA) [23], filters are removed from convolutional layers based on a redundancy analysis on the features maps, which aims to find, layer by layer, which filters can be removed without impacting on the accuracy metric.

Chapter 3

Materials

For this work, the CIFAR-10 and CIFAR-100 datasets are used [24]. These datasets are a subset of a broader collection of images made by researchers at the Massachusetts Institute of Technology (MIT) and New York University (NYU) in six months. They collected around 3 000 images for each non-abstract noun in the English dictionary by taking them from search engines like Google, Flick, and Altavista. They cleaned the data by removing perfect duplicates and those with many white pixels since they tend to be synthetic. This original dataset contained 80 million images, scaled to a 32×32 pixels resolution. The CIFAR-10 and CIFAR-100 are a subset of this dataset. Its images were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

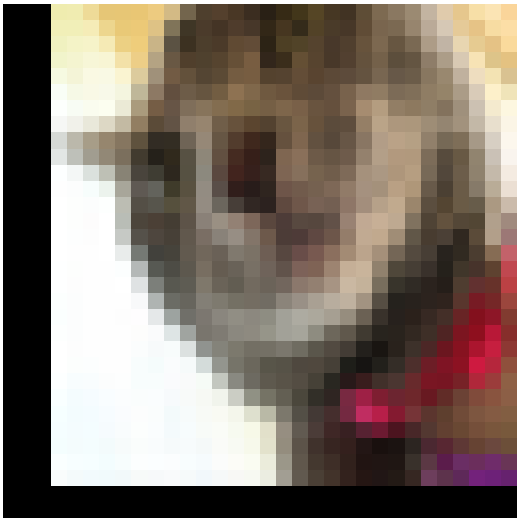

Figure 3.1: Image from the CIFAR-10 dataset, a cat.

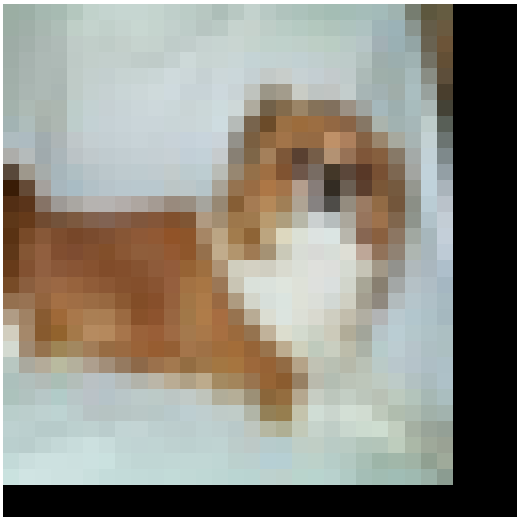

Figure 3.2: Image from the CIFAR-10 dataset, a dog.

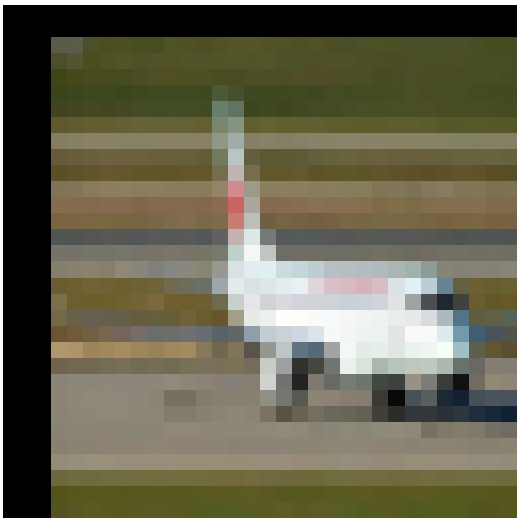

Figure 3.3: Image from the CIFAR-10 dataset, an aeroplane.

3.1 CIFAR-10

The CIFAR-10 dataset contains 60 000 images. These are divided into 10 classes with 6 000 images each. 50 000 for training, 10 000 for testing. The classes are: aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

The authors specify that the classes are mutually exclusive and that there is no overlap between categories. This is also true for automobiles and trucks: pickups are removed from this dataset.

Figure 3.4: Image from the CIFAR-10 dataset, a ship.

3.2 CIFAR-100

The CIFAR-100 dataset is similar to the previous dataset but it has 100 classes. Each class has 600 images: 500 for training and 100 for test. These categories are grouped into 20 super-classes divided as in table 3.1.

3.3 PyTorch

PyTorch [3] is a Deep Learning framework that focuses on both usability and speed. It is consistent with other scientific libraries and simple to debug while supporting GPU acceleration. Given this framework, we decided to use for the pre-trained model the ResNet-110 from [25] for the CIFAR-10 dataset and the ResNet-56[26] for CIFAR-100. These models have respectively 93.53 % and 72.63 % accuracy on their dataset.

3.4 ResNet

The ResNet [1] architecture is designed to solve the vanishing gradient problem when training deep neural networks. They introduced the "skipped connections" or "shortcut connections" to make the model learn the residual function: instead of learning

$$y = f(x), \tag{3.1}$$

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 3.1: Table of the CIFAR-100 dataset classes

it learns

$$y = f(x) + x. \tag{3.2}$$

Hence, $f(x)$ will yield the difference $y - x$. This name comes from fields like numerical analysis, where the term residual can refer to the error in an approximation of a solution.

These "skipped connections" make it easier for the model to learn identity mapping since it will be sufficient for the network to predict low values inside the residual block. They will be irrelevant once added to the input of the whole block, making it an identity map. These "shortcut connections" will help the gradient flow back to the network's top.

The ResNet-110 and ResNet-56 are models of 110 and 56 layers, respectively. These are variants of the general ResNet architecture proposed in the same paper. These are targeted for the CIFAR-10 and CIFAR-100 datasets.

The ResNet-110 has 54 Residual Blocks or Building Blocks; the ResNet-56 has 27. These are connected to the next one through a skipped and a standard connection. These blocks contain two convolutional layers in sequence.

Chapter 4

Methodology

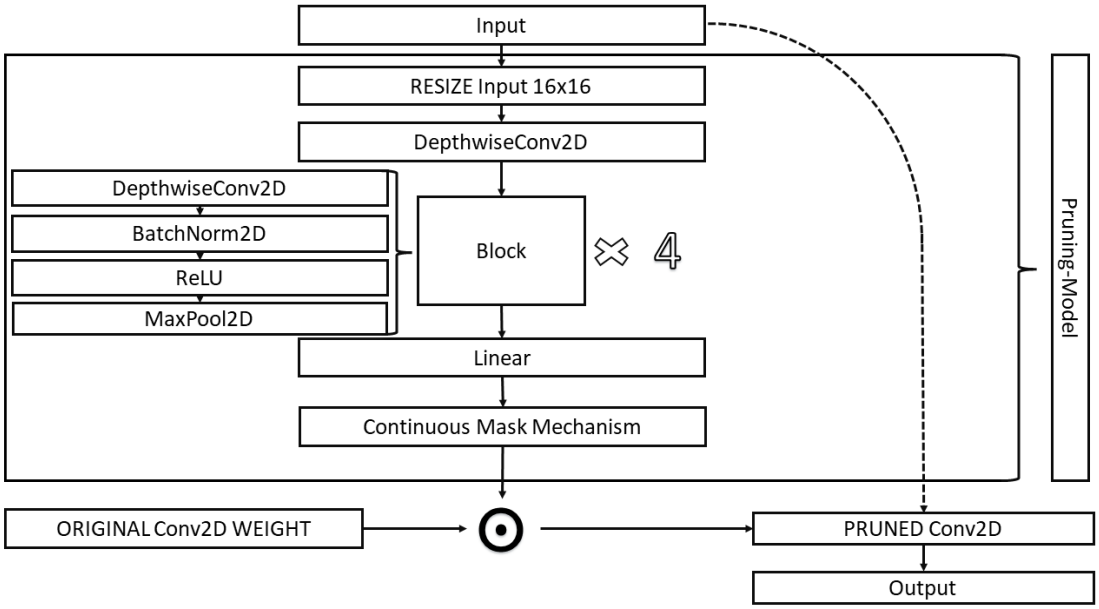

Figure 4.1: In this picture, we can see how our pruning-model, inside the PruningConv2D module, is structured.

This section is structured as follows:

4.1 The setting

As previously described, the ResNet-110 is the primary model, and the optimiser classically used on the CIFAR-10 and CIFAR-100 datasets is Stochastic Gradient Descent (SGD) [27]:

$$\theta_{t+1} = \theta_t - \eta \nabla J(L(\theta_t, x_i, y_i)), \tag{4.1}$$

where θ are the parameters, η is the learning rate, and ∇J is the gradient of the cost function L evaluated in θ_t for the i-th element of the dataset (x_i, y_i). In this work, we used its variant with the parameter weight decay set to 10^{-4} using the PyTorch implementation [28].

The batch size used is 128 and trained for 100 epochs; each epoch has 391 iterations for the training set and 79 for the validation set. In this work, we pruned in the first 4 000 iterations; the rest were left for fine-tuning. The learning rate starts at $1e-3$ for the ResNet-110 and $1e-2$ for the ResNet-56. Then, it decreases as follows: if the epoch is above 25 and the epoch is multiple of 10, then we multiply the original learning rate by $(0.941)^{(epoch-25)}$, so to reduce the final learning rate to approximately 1% of the initial learning rate.

We want to remark that we used only one loss for training both models involved. We will show how training both models was possible with the standard cross-entropy loss [29]:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \log(\hat{y}_{i,j}) \tag{4.2}$$

where $L(y, \hat{y})$ is the cross-entropy loss, N is the number of elements in the batch analysed, C the number of classes, y is the label, and $\hat{y}$ the predicted probability belonging to that class.

4.2 The classes

4.2.1 The OutputAnalyser class

The OutputAnalyser is a class that will be used to wrap the model. It is used to:

- replace the standard Conv2D layer with PruningConv2D module;

- train the new model;
- set the rules that determine the different pruning phases;
- keep the count of how many parameters have been pruned

We replace the Conv2D layers with the PruningConv2D module, which inherits from the PyTorch nn.Module. We iterate over the model layers to find those that are Conv2D. Once found, we replace them with the PruningConv2D module.

It is also used to train the model, which can be trained using the same "forward" method of the original object.

Through it, we can set the rules determining when the pruning phase will be alternated.

It also counts how many parameters have been pruned. The pruning is done inside the PruningConv2D module. We have to append the number of those still used to a list to keep track of all the pruned parameters. This list is an attribute of the OutputAnalyser class whose elements will be summed and saved in a temporary variable. This temporary variable will be printed to the terminal every 50 iteration.

4.2.2 The PruningConv2D module

At initialisation, the PruningConv2D class, whose mechanisms we see in figure 4.1, receives a model and the Conv2D module it will replace. The original weights of the convolutional layer will be stored inside this object as a PyTorch nn.Parameter [30] attribute. When pruning, the input of this layer is passed to the model's layer received when initialised.

The model we have chosen is composed of a resizing layer with a fixed input size: this to have height and width with the same cardinally through all the layers, thus avoiding discriminating the layers based on their input size. The size chosen is 16×16 pixels. Next, there is an input depth-wise convolutional [4] layer with input features equal to the ones of the module replaced. This layer's output size will be 2 times the one of the input. This is to have 2 features for each original characteristic extracted by the previous layer. Then, four blocks keep the number of features fixed. These blocks contain a convolutional layer whose number of groups equals the original input features dimension, a BatchNormlaisation [31] and a ReLU layer, and a MaxPooling layer [32] with kernel size equal to 2. Doing this will reduce the image to a single pixel with twice the initial number of channels. Finally, this output will be squeezed along the width and height dimensions since it is now reduced to a single pixel. Thus, only the batch and the feature dimensions are left. Then, this tensor is passed through a linear layer [33] with an output dimension equal to the original number of features.

Each layer has a different instance of this model, and the ensemble of these models will be referred to as the pruning-models in the following.

4.2.3 The Continuous-Mask Mechanism

The continuous-mask mechanism aims to get the current continuous-mask cm_t. We calculate the mean along the batch dimension - the $BM(\cdot)$ function - of the pruning model's output y. To this quantity, its mean along the batch and feature dimensions - the $M(\cdot)$ function - is subtracted. The resulting value will be divided by the square root of the variance of y; the $V(\cdot)$ function gets the variance of a tensor. The previous continuous-mask cm_{t-1} and this quantity are then halved to weight them equally and added. To this result, we add an *offset* variable, equal to 1.15 in our setting, and subtract the sum of the current iteration - *iter* - with the *interacting* variable, which consists of a value that the user can change while the pruning is being executed and that will be better explained later, divided by the maximal number of pruning iteration, *max_iter*. The result of all of these operations is passed through a ReLU activation function to let the negative values be zeros. The resulting formula is the following:

$$cm_t = ReLU\left(\frac{cm_{t-1}}{2} + \frac{BM(y) - M(y)}{2 * \sqrt{V(y)}} + \left(offset - \left(\frac{iter + interacting}{max\_iter}\right)\right)\right) \quad (4.3)$$

Containing only zeros or positive real numbers, cm_t values belong to $\mathbb{R}^+$.

4.3 Training Phases

The training of this model requires, or at least supports, two different training phases.

4.3.1 Pruning Phase

The pruning phase lasts for 4 000 iterations. In this phase, both the model and the pruning-models are trained.

4.3.2 Fine-Tuning

After the pruning, we can start the fine-tuning phase. During this, the layers will not be able to prune anymore. This step is necessary to reduce network error and to bring the model back to convergence.

4.4 Interactive Pruning

As a plus of this approach, since we are subtracting a scalar in equation 4.3, we can interact with this scalar variable by increasing or decreasing it, thus pruning

more or fewer parameters, respectively. This is useful to obtain a target pruning percentage without finding the right balance before starting the program.

4.5 Obtaining the Pruned Model

To prune the ResNet model, the following accoutrements must be considered.

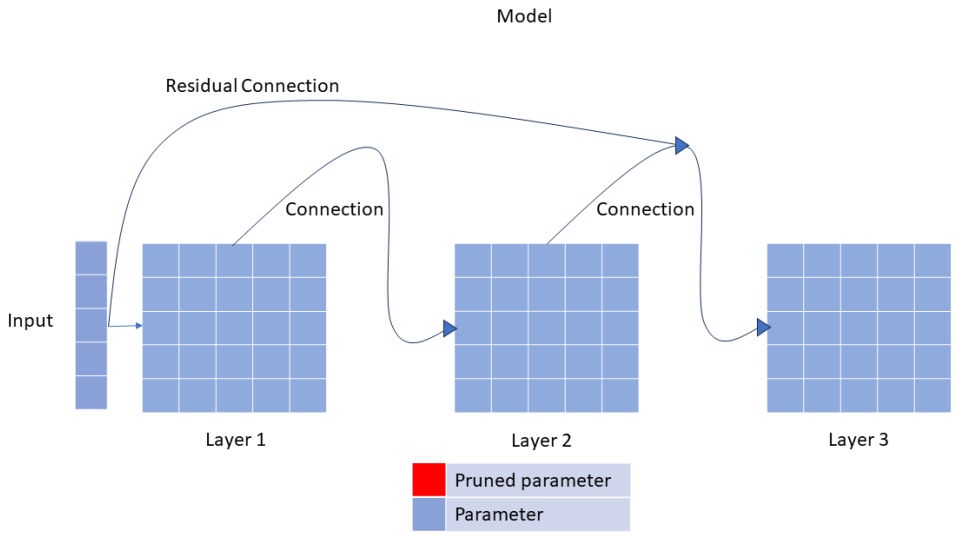

Figure 4.2: A model made of a residual connection before pruning.

First of all, since we are pruning the input channel of the following layer based on the prediction of the previous one, we also must prune the previous layer's output accordingly. For example, if we are removing the 2^{nd} and the 4^{th} input channels from the 2^{nd} layer, we must also remove them from the first's output channels.

Secondly, we have to take into account the presence of residual connections. In this model, there is a residual connection every two layers. That connection sums the first's input with the second's output. Hence, the same channels have to be pruned for both these tensors: if we are removing the 3^{rd} and the 4^{th} from the input of the 1^{st} layer, we must also remove them from the output of the 2^{nd} and the input of the 3^{rd}, which is the one that receives the residual input.

Since there are no layers between two subsequent residual connections, this schema must be applied along all the 54 residual blocks of the ResNet-110 and the 27 of the ResNet-56. This introduces more complexity in how the pruning choices must be done. To choose a shared continuous-mask for these layers, we averaged

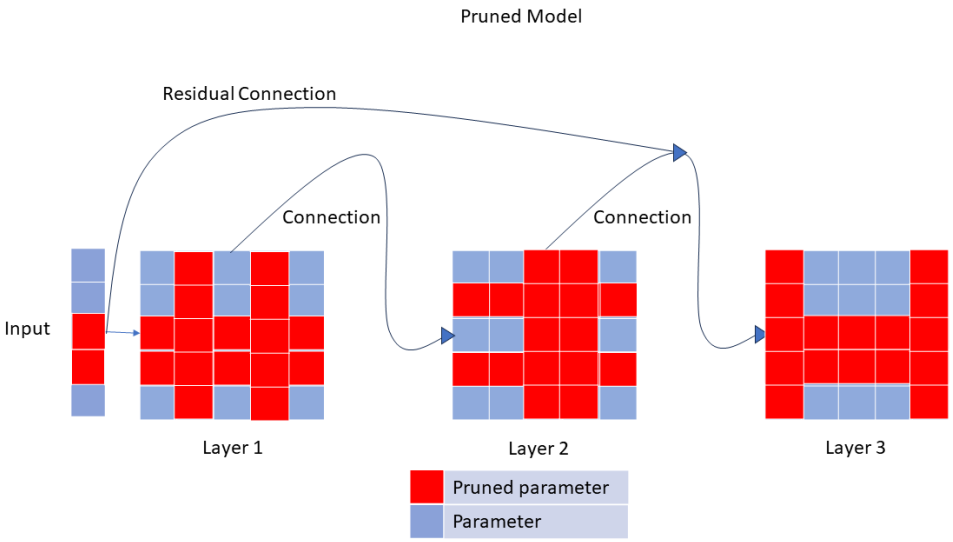

Figure 4.3: A model made of a residual connection after pruning.

them and subtracted a value to reintroduce zeros in the final mask: we subtracted the 70 % of the mean of all the continuous masks generated by these layers and then replaced them with this new one.

Another step in pruning this model's architecture is considering the presence of layers that change the number of output channels while keeping the residual structure. For example, we have a layer that doubles the number of features from 32 to 64. This layer's output passes through another layer and is added to the original input with 32 channels. In the non-pruned model, it was done by zero-padding: 16 channels with zero values were concatenated before and after the original input, thus obtaining 64 channels. Since the number of dimensions is variable, we can neither double it nor rely on the assumption that they are contiguous since some may have been pruned. To solve this problem, we filled a tensor with the initial input shape of 32 channels with zeros. We replace the zeros of this tensor with the pruned input by applying the corresponding mask. We then do the zero-padding step to double the number of channels. Now, we mask this tensor with the following layer's input pruning-mask. This way, we can safely add it to its residual counterpart without mixing the channel's features of the two branches of the skipped connection.

Lastly, the batch normalisation layers following the convolutional layers must be pruned according to the previous layer's output.

Once all of this has been done, we can replace the PruningConv2D modules

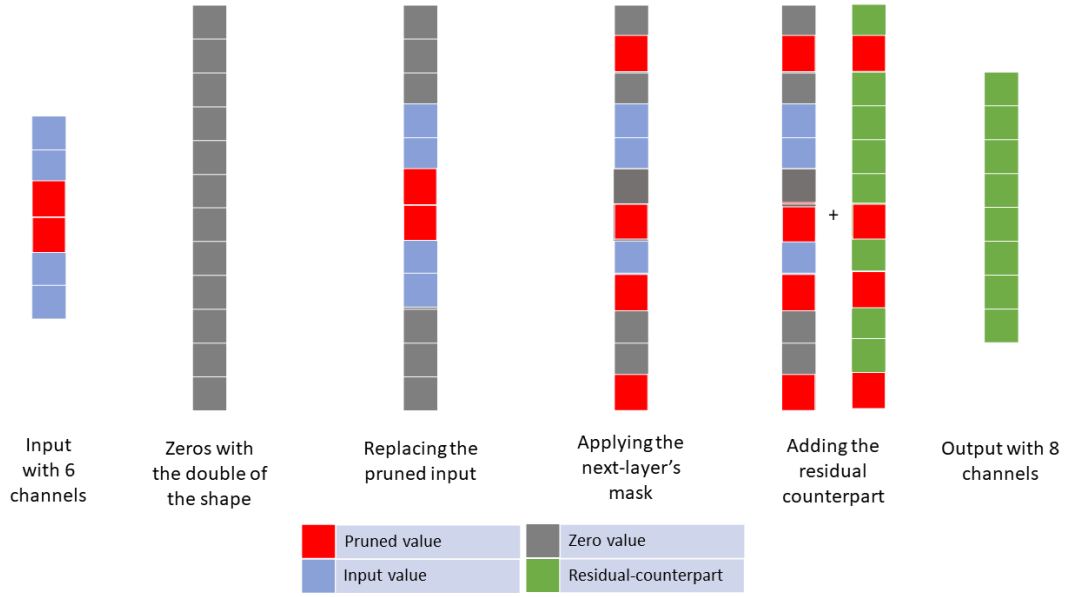

Figure 4.4: From left to right, we can see the pruning process when different channel sizes are involved.

with the now pruned Conv2D.

Figures 4.2, 4.3 and 4.4 show these steps.

Chapter 5

Results

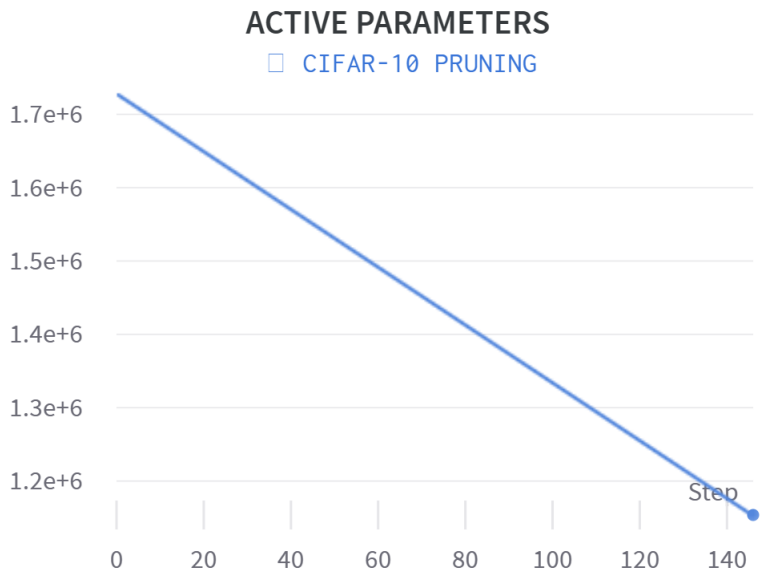

Figure 5.1: Number of pruned parameters before and after the pruning phase on the CIFAR-10 dataset.

We report the number of active parameters in figure 5.1, the FLOPs in figure 5.2, the trend of the pruned paramters during training in figure 5.3, the accuracy trend in figure 5.5, the loss trend in figure 5.4, and the learning rate in figure 5.6 along the training on the CIFAR-10 dataset. In picture 5.5, 5.4 and 5.3, we can see the alternating of the pruning phases: we first have a decrease in accuracy and a peak in the loss when pruning more and more parameters. Then, a recovery in accuracy, decreased loss and stagnation in pruning during the fine-tuning phase.

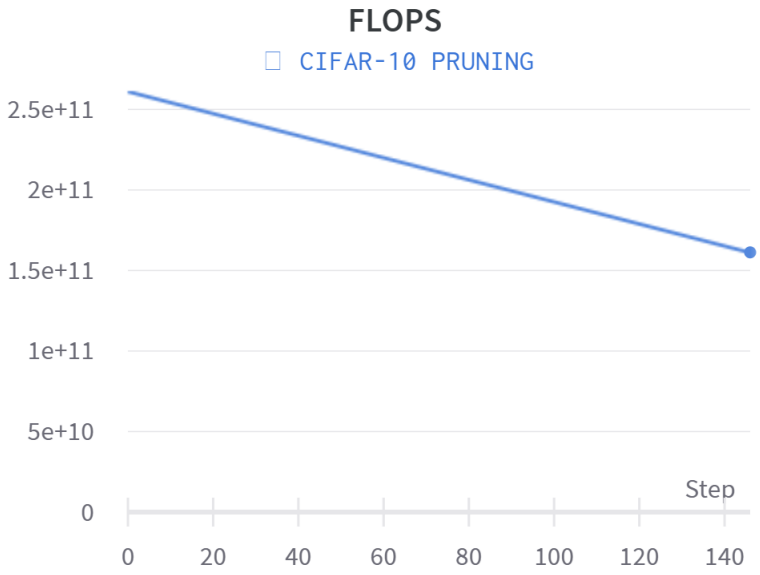

Figure 5.2: Number of FLOPS before and after the pruning phase on the CIFAR-10 dataset.

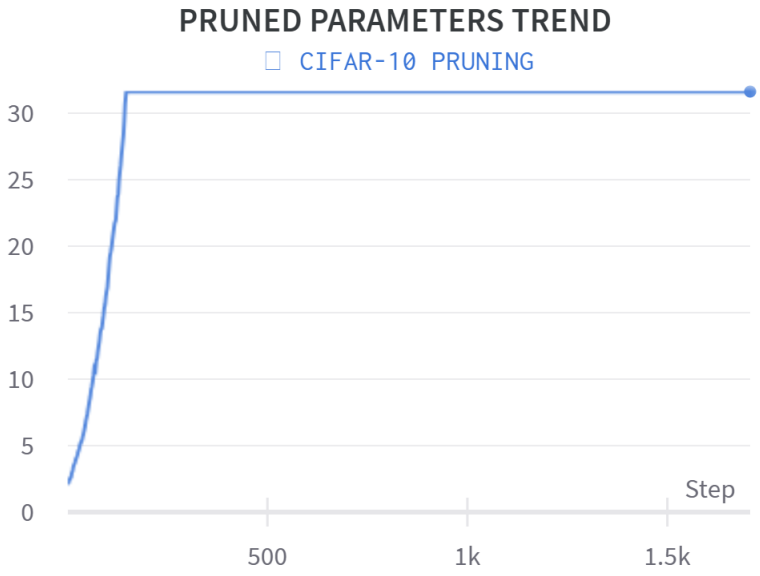

Figure 5.3: The pruned parameters trend during training on the CIFAR-10 dataset, every step equals 50 iterations.

Figure 5.4: The loss trend during training on the CIFAR-10 dataset, every step equals 50 iterations.

In table 5.1, we report the results. Here, we can see that this may be a promising technique. The missing *Pruned Parameters* values marked as not available (*NA*) were not reported in the original papers.

Experiment	Error %	Pruned Parameters %	Pruned FLOPs %
Original	6.47	0.0	0.0
PFEC [8]	6.70	32.4	38.6
FPGM [10]	6.26	NA	52.3
SFP [9]	6.14	NA	40.8
HRank [14]	6.64	59.2	58.2
NISP [18]	6.65	43.25	43.78
Ours	7.16	33.23	38.23

Table 5.1: Results comparison on CIFAR-10 of the pruning of the ResNet-110 with 1.72×10^6 initial parameters. Among the papers cited in section 2, only those with the results on the CIFAR-10 dataset concerning the ResNet-110 are reported.

We can see in 5.2 that, although the error increased from 27.37 % to 29.13 %, we removed 26 % of the floating point operations per second (FLOPs) and 10 % of the parameters of the ResNet-56, on a hundred classes dataset.

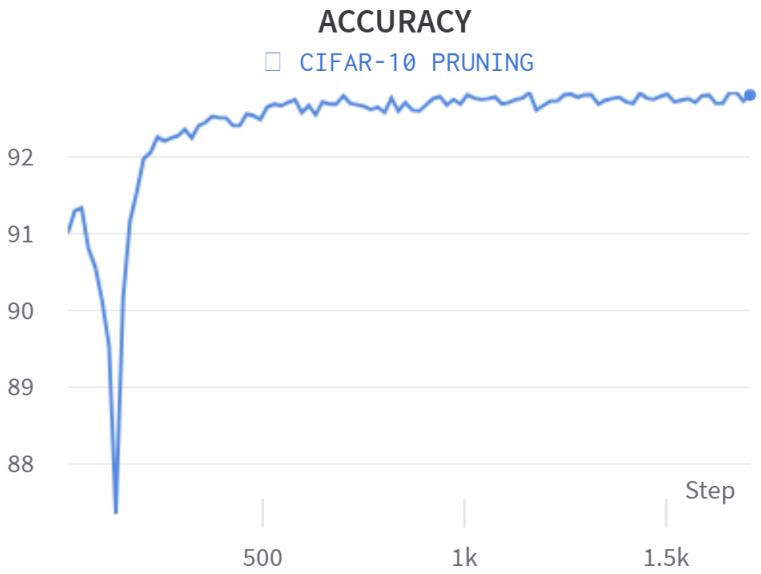

Figure 5.5: The accuracy trend during training on the CIFAR-10 dataset, we calculate and report the accuracy score on the validation set after each training epoch.

Experiment	Error %	Pruned Parameters %	Pruned FLOPs %
Original	27.37	0.0	0.0
FMA [23]	29.20	43.02	35.25
Ours	29.13	10.76	26.1

Table 5.2: Result on CIFAR-100 of the pruning of the ResNet-56 with 0.86×10^6 initial parameters.

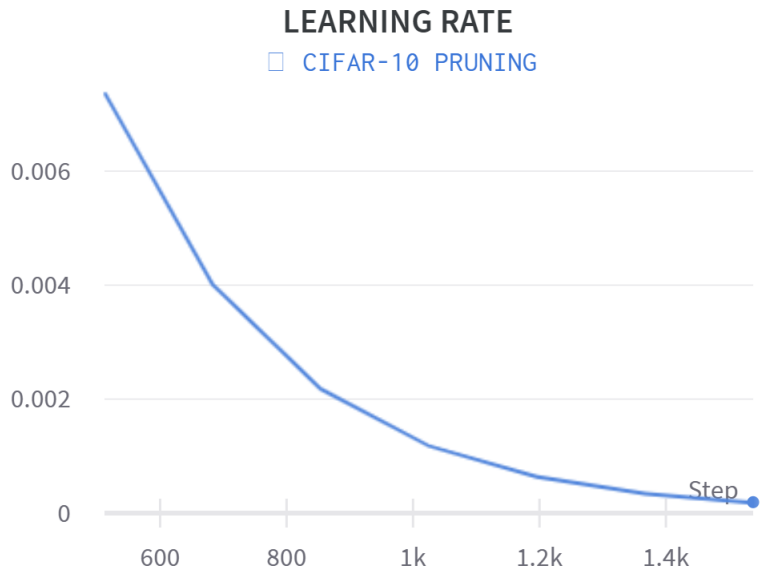

Figure 5.6: The learning rate trend during training on the CIFAR-10 dataset, every step equals 50 iterations.

Chapter 6

Discussion

The main focus of this work is to apply structured pruning, not relying on a fixed criterion but on external Deep Learning models. Moreover, this method could be extended to frameworks different from PyTorch, other tasks and kinds of layers.

We have used pruning-models: one for each convolutional layer. These models' predictions are exploited to do the pruning. They analyse the previous layer's output feature map to prune the previous layer's output dimension and the following layer's input dimension. Given that the ResNet model's architecture is based on the residual connection, we must consider them and prune them according to the rules explained in the Methodology. These rules are specific to the architecture but must be considered: the predictions of these layers are average and shared. Moreover, by looking at the formula of the continuous-mask mechanism 4.3, we can see that they are weighted with new predictions. The way we have implemented the continuous-mask mechanism allows the gradient to flow through the model with no need for approximations. In this way, we do not need an external and hard-to-balance loss, and we have a method that is agnostic to the task since we can use the original loss.

Once pruning is done, we can merge the last prediction of the pruning-models to do the pruning: the predictions are first multiplied by the layer's parameters; then, the channels with zero values are removed from the network. Finally, the PruningConv2D are replaced with the now pruned Conv2D. This way, the model will not have extra computations: neither the pruning-models nor the masks are kept. Given that the channels are removed, we can see an increase in the inference speed by looking at the FLOPs count percentage decrease. Deploying it does not require other optimisation steps: the network's tensors and parameters are dense. In an unstructured pruning setting, they would have been sparse, thus requiring specific hardware capable of handling this type of data.

6.1 About methods

We started from the piggyback [34] method, which uses parameters as gates to turn on and off parameters. Although not meant for pruning, we thought this idea could be useful in our use case. A problem with this method was that the gradient needed to be approximated: the gates, on or off, are not differentiable. Thus, we decided to replace them with a ReLU activation function. The ReLU activation function is often used to create non-linearity in the model. In our case, it selects which parameters to keep and discard.

The idea of using a model to do the pruning came from an attempt we made to do a sort of knowledge distillation with a meta-model: the aim was to predict the parameter's value of a student model by training a meta-model to fill them starting from the ones of trained teacher model. This approach required a lot of resources since the meta-model should be, we thought, bigger than the teacher and the student model. We reformulated the problem from the knowledge distillation to pruning settings, not to have the model predict the parameters' value directly, but which to keep and discard. This approach led to significantly better results than those we first obtained with the meta-model method, for which we do not report the scores obtained.

6.2 About results

This method seems promising since the results obtained are close to the state-of-the-art. Further studies may exploit the residual connections to build blocks of layers in a ResNet-like style for the pruning-models, thus making them deeper and able to extract more abstract features.

Chapter 7

Conclusion

This work proposes an approach to prune Convolutional layers on the ResNet model through external Deep Learning models. Specifically:

- this method does not choose in advance the criterion, but the pruning-models themselves select the channels which are less relevant for the predictions;
- the gradient flows to the models without needing another hard-to-balance loss function;
- we have chosen the classification task, but given that our method is not task-specific, it may be used in other settings;
- we have applied our pruning algorithm to the Conv2D layer implementation made by PyTorch, but our method could be extended to other layers and frameworks;
- the number of pruned parameters can be influenced before and during training, allowing one to reach a target model compression ratio;
- we implemented an architecture for the pruning-models: further studies may explore different configurations and find a better-suiting one by widening, deepening or changing this model's structure;
- some masks, such as the ones between residual layers, must be shared to apply the pruning. Finding an architecture that minimises the number of shared masks may help further increase the model compression ratio and performance.

With this methodology, we got close to the state-of-the-art on the CIFAR-10 dataset, where we removed 33.23 % of parameters and reduced the FLOPs by 38.23 %, making the error increase by 0.69 %. On the CIFAR-100 dataset, we

removed the 10.76 % of the parameters and reduced the FLOPs by 26.1 %; the error increased by 1.76 %.

Bibliography

[1] 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* (2016), pp. 770–778 (cit. on pp. 9, 21).

[2] Solomon Kullback and Richard A Leibler. «On information and sufficiency». In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86 (cit. on p. 10).

[3] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Proceedings of the Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035 (cit. on pp. 11, 21).

[4] *torch.nn.Conv2d*. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html. Accessed: [26/08/2023] (cit. on pp. 11, 25).

[5] Jonathan Frankle and Michael Carbin. «The lottery ticket hypothesis: Finding sparse, trainable neural networks». In: *International Conference on Learning Representations*. 2019 (cit. on p. 13).

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: a simple way to prevent neural networks from overfitting». In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on p. 13).

[7] Yang He and Lingao Xiao. «Structured Pruning for Deep Convolutional Neural Networks: A survey». In: *arXiv preprint arXiv:2303.00566* (2023) (cit. on pp. 13, 18).

[8] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. «Pruning Filters for Efficient ConvNets». In: *arXiv preprint arXiv:1608.08710* (2016) (cit. on pp. 14, 32).

[9] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. «Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks». In: *arXiv preprint arXiv:1808.06866* (2018) (cit. on pp. 14, 32).

[10] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. «Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration». In: *arXiv preprint arXiv:1811.00250* (2018) (cit. on pp. 14, 32).

[11] Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. «COP: Customized Deep Model Compression via Regularized Correlation-based Filter-Level Pruning». In: *arXiv preprint arXiv:1906.10337* (2019) (cit. on p. 14).

[12] Karl Pearson. «Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity, and Panmixia». In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 187 (1896), pp. 253–318 (cit. on p. 15).

[13] Yihui He, Xiangyu Zhang, and Jian Sun. «Channel pruning for accelerating very deep neural networks». In: *Proceedings of the International Conference on Computer Vision*. 2017, pp. 1389–1397 (cit. on p. 15).

[14] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. *HRank: Filter pruning using high-rank feature map.* arXiv preprint arXiv:2002.10179v2 [cs.CV]. 2020 (cit. on pp. 16, 32).

[15] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. «ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression». In: *arXiv preprint arXiv:1707.06342* (2017) (cit. on p. 16).

[16] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. «Runtime neural pruning». In: *Proc. Adv. Neural Inform. Process. Syst.* Vol. 30. 2017 (cit. on p. 16).

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation.* Tech. rep. Institute for Cognitive Science, University of California, 1985 (cit. on p. 16).

[18] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. «NISP: Pruning Networks using Neuron Importance Score Propagation». In: *arXiv preprint arXiv:1711.05908* (2018) (cit. on pp. 16, 32).

[19] Z Liu, J Li, Z Shen, G Huang, S Yan, and C Zhang. «Learning efficient convolutional networks through network slimming». In: *Proceedings of the International Conference on Computer Vision*. 2017, pp. 2736–2744 (cit. on p. 17).

[20] Z Huang and N Wang.
«Data-driven sparse structure selection for deep neural networks».
In: *Proceedings of the European Conference on Computer Vision*. 2018, pp. 304–320 (cit. on p. 17).

[21] W Wen, C Wu, Y Wang, Y Chen, and H Li.
«Learning structured sparsity in deep neural networks».
In: *Proceedings of the Advances in Neural Information Processing Systems*. 2016, pp. 2082–2090 (cit. on p. 17).

[22] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang.
«Soft filter pruning for accelerating deep convolutional neural networks». In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 2018, pp. 2234–2240 (cit. on p. 18).

[23] Qi Li, Hengyi Li, and Lin Meng. «Feature Map Analysis-Based Dynamic CNN Pruning and the Acceleration on FPGAs».
In: *Electronics* 11.18 (2022), p. 2887. DOI: 10.3390/electronics11182887 (cit. on pp. 18, 33).

[24] Alex Krizhevsky. *Learning multiple layers of features from tiny images*.
Tech. rep. University of Toronto, 2009 (cit. on p. 19).

[25] Yerlan Idelbayev.
Proper ResNet Implementation for CIFAR10/CIFAR100 in PyTorch.
https://github.com/akamaster/pytorch_resnet_cifar10.
Accessed: 23/09/2023 (cit. on p. 21).

[26] Chen Yao.
pytorch-cifar-models: Pretrained models on CIFAR10/100 in PyTorch.
https://github.com/chenyaofo/pytorch-cifar-models.
Accessed: 23/09/2023 (cit. on p. 21).

[27] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller.
«Efficient backprop». In: *Neural networks: Tricks of the trade*. Springer. 1998, pp. 9–48 (cit. on p. 24).

[28] *torch.optim.SGD*.
https://pytorch.org/docs/stable/generated/torch.optim.SGD.html.
Accessed: [25/09/2023] (cit. on p. 24).

[29] Claude E Shannon. «A mathematical theory of communication».
In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423 (cit. on p. 24).

[30] *torch.nn.Parameter*.
https://pytorch.org/docs/stable/generated/torch.nn.parameter.Parameter.html
Accessed: [11/09/2023] (cit. on p. 25).

[31] *torch.nn.BatchNorm2d*.
https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html.
Accessed: [08/09/2023] (cit. on p. 25).

[32] *torch.nn.MaxPool2d*.
https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html.
Accessed: [27/08/2023] (cit. on p. 25).

[33] *torch.nn.Linear*.
https://pytorch.org/docs/stable/generated/torch.nn.Linear.html.
Accessed: [26/08/2023] (cit. on p. 25).

[34] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. «Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights».
In: *Proceedings of the European Conference on Computer Vision (ECCV)*.
2018, pp. 72–88 (cit. on p. 36).