Hasso Plattner Institute
Chair for Data Engineering Systems

Master Thesis

Benchmarking Pruning and Quantization Techniques

Benchmarking von Pruning- und Quantisierungstechniken

Jonas Schulze

Time frame: May 27, 2024 - November 27, 2024

First Supervisor
Prof. Dr. Tilmann Rabl
Chair for Data Engineering Systems

Second Supervisor
Prof. Dr. Gerard de Melo
Chair for Artificial Intelligence and Intelligent Systems

Advisor
Nils Straßenburg
Chair for Data Engineering Systems

Acknowledgements

I would like to thank my supervisors Prof. Dr. Tilmann Rabl and Nils Straßenburg for their great support and guidance throughout my work on this Master thesis. The enjoyable working atmosphere and available assistance whenever needed greatly facilitated the work. The cooperation, not only during this thesis but the many tutoring opportunities throughout my studies, have been more than a pleasure.

Furthermore, I would like to thank my family, especially my parents and great parents, for their help and assistance in the writing of my thesis and the past six and a half years of university.

And of course, thanks to all my closest friends, for they stood by my side each and every day. I received a great amount of encouragement and emotional assistance through their support. On that note, I would like to thank and say goodbye with a heavy heart to my dearest friend Marcel, who just passed away as I am writing the finishing touches of my thesis. You will be deeply missed, but our memories won't be forgotten.

Abstract

Deep learning models feature a great amount of parameters, i.e. nodes and weights, which grant these architectures their name. But, sometimes you need to reduce the model size and simplify the neural network, e.g. when limited by hardware constraints or emphasis lies on energy efficiency. Faster inference is a more desirable model trait than top end accuracy in this setting. Model compression like pruning and quantization act as a form of regularization on the neural network and help achieve efficiency.

In this master thesis, we investigate the effect of pruning and quantization techniques on different models and determine which compression tool has the greatest effects in terms of model size reduction and inference speed gain while preventing an impactful loss in accuracy. Our focus is on an as out-of-the-box usage of the tools as possible, with application on pretrained models. We build a framework for benchmarking which analyzes models and compression techniques of a user.

We find out that some pruning and quantization tools do not match their expectations in our setting and are associated with a great amount of work to set up. But, some approaches find success on models which are thought to be optimized and compressed to the limit due to their deployment on edge devices.

Zusammenfassung

Deep Learning Modelle verfügen über eine große Anzahl von Parametern, also Knoten und Gewichte, die diesen tiefen Architekturen ihren Namen geben. Manchmal ist es jedoch notwendig, die Größe des Modells zu verringern und das neuronale Netz zu vereinfachen, z. B. im Rahmen von Hardwarebeschränkungen oder sobald man Wert auf Energieeffizienz legen möchte. In diesem Fall ist eine schnellere Inferenz eine wünschenswertere Eigenschaft des Modells als die maximale Genauigkeit. Hierbei helfen Modellkomprimierungen wie Pruning und Quantisierung. Sie wirken als eine Form der Regularisierung des neuronalen Netzes.

In dieser Masterarbeit untersuchen wir die Auswirkungen von Pruning- und Quantisierungstechniken auf verschiedene Modelle. Wir ermitteln, welches Komprimierungswerkzeug den größten Einfluss in Bezug auf die Reduzierung der Modellgröße und der Latenz während der Inferenz hat und inwieweit gleichzeitig ein erheblicher Genauigkeitsverlust verhindert wird. Unser Schwerpunkt liegt dabei auf einer möglichst unkomplizierten Anwendung auf vortrainierten Modellen. Wir entwickeln eine Rahmenstruktur für das Benchmarking, welches zur Analyse eigener Modelle und Komprimierungstechniken eines Benutzers verwendet werden kann.

Wir finden heraus, dass einige Pruning- und Quantisierungswerkzeuge in unserem Versuchsaufbau nicht den Erwartungen entsprechen und mit einem hohen Einrichtungsaufwand verbunden sind. Einige Ansätze haben dafür jedoch Erfolg bei Modellen, von denen man annimmt, sie seien aufgrund ihres Einsatzes auf Edge-Geräten bis an die Grenze optimiert und komprimiert.

Contents

1 Introduction

Pruning and quantization are essential techniques in the field of machine learning and play a pivotal role for their deployment [4, 31, 43]. When implementing a model, we often do not have access to the resources that have been used during development and training and are working in a resource constrained environment. We have to make use of the limited assets at hand. Pruning and quantization helps us in this regard as they trade off a portion of performance to save a significant amount of memory, time, or power usage for a more efficient operation. These techniques foster generalization and robustness, which lead to a better inference. The pruned models are more accessible, scalable, and energy efficient than the base model. The benefits of a successful model down scaling are alluring, and there are plenty of techniques to be used.

Some papers cover only a single compression technique or limited amount of models and benchmarking criteria [3, 8, 14]. AlexNet, ResNet and VGG are the first choices for benchmarking a new pruning strategy with little regard to more modern architectures [5, 49]. Benchmarking papers do not incorporate a central framework to make ease of equivalent matching against each other. So far, there is no benchmarking standard and a lack of comparability [5]. Providing such a tool additionally supports reproducibility and effortless execution. A developer does not need to implement several pruning and quantization techniques from scratch to evaluate them, but simply uses them out-of-the-box and applies them as easily as possible.

The focus of this master thesis is the benchmarking and comparison of two of the most common model compression techniques, pruning and quantization. We evaluate their effectiveness on a variety of models. Here, we emphasize out-of-the-box usage of each part of our framework and the application on a pretrained model of a potential user.

Contributions This thesis offers the following contributions:

1. We develop a benchmarking tool for measuring the performance of pruning and quantization techniques on pretrained models.

2. We incorporate our tools out-of-the-box to render model compression as accessible as possible.

3. We run benchmarks on our collection of models, datasets, pruners and quantizers.

4. We investigate and discuss the findings from our benchmark runs in regard to accuracy performance, inference speed and model size.

5. We offer custom runs with our benchmarking backend as an option to extend our framework.

Thesis Structure In Section 2 we cover related work in the field of model compression, pruning and quantization.

Section 3 introduces relevant concepts of this thesis and covers the foundations of pruning and quantization and their requirements.

Section 4 gives an overview of our framework and how to use our benchmarking tool.

In Section 5 we investigate our findings regarding the impact of model compression on accuracy, inference speed and model size. We show results of combining pruning and quantization and our application on a transformer architecture.

Section 6 concludes our work and offers discussion on our findings, and in Section 7 we give an outlook on future work on this topic.

2 Related Work

In this section, we discuss related work in the field of pruning and quantization and especially the benchmarking of these compression techniques.

Pruning and quantization techniques offer an efficient variant of a given base model. It is possible to remove 90 percent of a model's weights and barely notice a drop-off in accuracy [52]. Different pruning techniques have been analyzed and compared to one another in studies and surveys [2, 5] in which they measure pruning algorithms mostly by their performance in regard to accuracy, while the lack of a standardized benchmarking tool is discussed as well. They investigate the advantages and disadvantages of pruning models in contrast to designing the base model small but dense [52].

Other papers introduce and evaluate quantization techniques [37, 41] reporting competitive accuracy results with as low as four bits [8] and one bit [22] integer quantization with studies measuring the reduction of memory footprint and latency up to a factor of 16x [16].

There is little comparison between these two in literature [29, 31]. These papers investigate the effectiveness of pruning and quantization on their own and in combination, usually in regard to accuracy. While both have their use cases, quantization generally outperforms pruning [29].

Liang et al. [31] analyze multiple pruning and quantization algorithms and libraries when applied to common networks. They have interesting findings regarding both techniques and supply advices and best practices alongside. But, the results are not accompanied by source code or a general framework. The paper focuses on accuracy with little to no regard for inference speed benchmarks and changes in model size.

Augasta et al. [2] is a comparative study for pruning techniques as well. They categorize the methods and pick a representative algorithm for comparison on four different image classification tasks. But, similar to Liang et al., their theoretical findings offer no tool for benchmarking the pruning tasks. They do not cover quantization as a compression technique besides pruning and base their results only off accuracy preservation and not inference time and model size reduction.

Blalock et al. [5] provide a meta analysis of over 80 papers and 100 pruned model experiments. Similar to our work, they approach the problem of papers rarely reporting controlled comparisons of their models. They highlight pitfalls in literature and offer best practices for avoiding them. Alongside their research, they developed their own benchmarking framework, ShrinkBench [6], based on PyTorch. Using predefined and standardized models and datasets, it offers training and testing, pruning

and fine-tuning of the model, metric calculations and plotting. For evaluation, they offer a scoring function for models. ShrinkBench is, to our current knowledge, the closest related work to our framework. But, they do not cover quantization and have a sole focus on pruning. We offer more variety in models and cover the measurement of inference time.

3 Background

In this section, we cover the basic concepts required for understanding the contents of the following thesis. In Subsection 3.1 we explain the basic terminology around Artificial Intelligence. We focus on Convolutional Neural Networks in this thesis, which we define in Subsection 3.2. In Subsections 3.3 and 3.4 we cover pruning and quantization as our compression techniques and discuss their hardware support in Subsection 3.5.

3.1 Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence (AI) describes machines, usually computer systems, simulating intelligent behavior. Similar to natural intelligence exerted by humans, these applications learn from and interpret their environment, which helps them process input data and inherit problem-solving abilities [10, 11].

Machine Learning (ML) is a subset of AI. While AI is a rather broad term for computer systems mimicking human behavior, ML is the branch of AI that develops algorithms and statistical models to tackle complex tasks without being explicitly coded to. They learn how to solve them through inference and pattern recognition acquired through processing usually large amounts of historical data [1, 10].

Deep Learning (DL) is a subset of ML [10]. The term 'deep' refers to the depth of the multi layered neural networks mimicking its neuroscientific counterpart, the human brain. The neurons of a neural network are the more abstract and mathematical pendant of biological neurons.

A neural network is an executable directed acyclic graph (DAG) consisting of layers of neurons. A neuron receives inputs $x_1, x_2, ..., x_n$ from the previous layer of n neurons (or the input data in the case of the first layer). Each input x_a from a previous neuron a is associated with a weight w_a which controls the importance of the specific input for the current neuron. The neuron computes a weighted sum of the inputs, $z = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + b$ with b being the bias used for shifting the function. The sum z is the input for an activation function (e.g. Sigmoid, ReLU, Softmax). The non-linearity of such an activation function allows deep neural networks to express complex relationships of the data. The result of the function is fed as input into the next layer of neurons (or represents the output in case of the last layer). Usually every neuron is connected to every neuron of the previous and upcoming layer, hence this architecture is called a fully connected neural network.

In Figure 1 we gain an overview of the architecture of a neural network. Any layer besides the first and last are called hidden layers, which grant neural networks

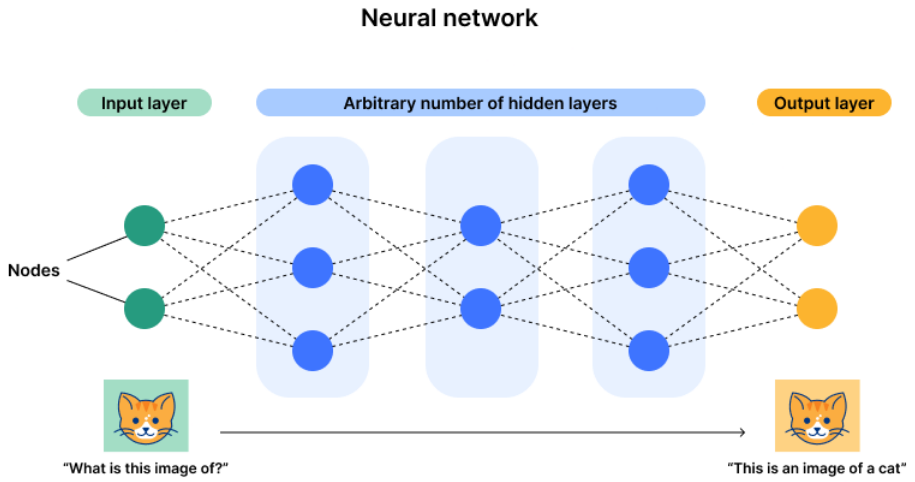


Figure 1: Sketch of the architecture of a neural network. The network is trained upon a specific task, e.g. image classification in this example. The first layer receives the input data i.e. an image, the hidden layers perform calculations according to previously learned patterns which lead to a decision output in the last layer of the network.

Source: Cloudflare [9]

their processing power [12]. The mathematical framework that is the neural network is also what is commonly called the **model**.

3.2 Convolutional Neural Networks

In this thesis we focus mostly on **Convolutional Neural Networks (CNN)** which are a type of architecture in the field of deep learning. They process image data as input, which are two-dimensional in nature (excluding RGB channels for colored images). In addition to fully connected layers, CNNs include convolutional layers. These feature filters also known as kernels. Kernels are matrices of weights $w_{11}, w_{12}, ..., w_{kk}$ for a kernel of size $k * k$. Contrary to fully connected layers, weights of a kernel do not multiply with each input $x_{11}, x_{12}, ..., x_{nn}$. The kernel multiplies with as many weights as the kernel's shape allows and 'slides' to a new set of inputs and applies the same weights.

We stitch each computation's output together to form the input for the next layer, as examined in Figure 2. Output y_{11} is the result of $w_{11} * x_{11} + w_{12} * x_{12} + w_{21} * x_{21} + w_{22} * x_{22}$. The kernel slides onto the next region of input, e.g. the quadrant of x_{12}, x_{13}, x_{22} and x_{23} and computes $y_{12} = w_{11} * x_{12} + w_{12} * x_{13} + w_{21} * x_{22} + w_{22} * x_{23}$ with the same kernel weights as before. This architecture allows each kernel to recognize patterns in different regions of the input [44].

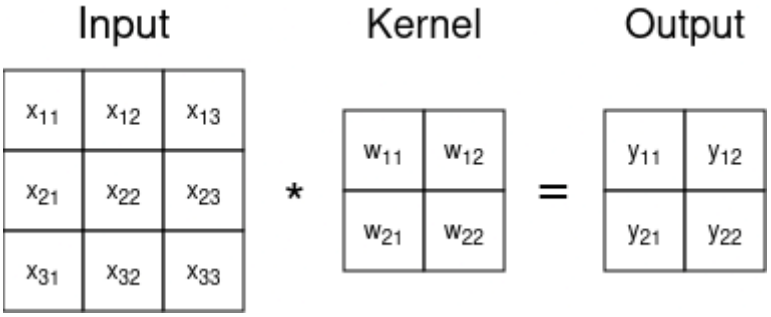


Figure 2: Sketch of the architecture of a convolutional layer. The entries of the kernel are being multiplied with regions of the input. As the kernel is usually smaller, it 'slides' across the larger input to cover each region. Each set of products sums up to form the output.

3.3 Pruning

The goal of pruning is to produce efficient models without losing too much accuracy. It regulates a neural network and reduces its complexity by removing parts of the model (typically neurons or weights) that are deemed unnecessary or redundant. The removal makes models smaller and simple, usually resulting in faster and slimmer models with less memory consumption and lower latency. The challenge is to find the sweet spot of efficiency gain and loss in performance.

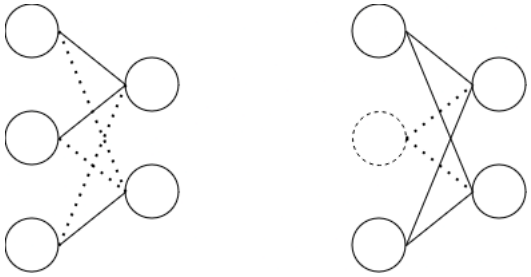


Figure 3: Unstructured pruning (left) does not value the integrity of the model, whilst structured pruning (right) maintains connectivity of the network by removing neurons and groups of weights.

In pruning, we categorize techniques in two different groups, structured and unstructured pruning. **Structured pruning** prunes groups of weights that structurally belong together, such as filters or channels. This leads to preserving the overall integrity of the model while altering input and output shapes of the layers, as seen in Figure 3.

In contrast, **unstructured pruning** removes individual parameters from the model, without a specific pattern. The resulting model loses its structural integrity, but is less restricted on the pruner's choice of weight removal [24, 34].

We plan on using the following pruning techniques, with focus on application in a computer vision setting. They are, to minimize overhead, all part of the PyTorch toolbox.

Random Pruning One of the simplest forms of a pruning technique is random pruning. Here, we prune a set fraction of weights from the model by random. For example, if we apply random pruning with a degree of 0.5, we randomly choose half of the pretrained weights and set them to zero. The nature of the technique does not take the significance of a weight for inference into account.

L1 Pruning (Unstructured) L1 pruning calculates importance scores of weights based on their L1 norm and zeros accordingly. We get a model with more relevant weights left intact than with a random selection [15].

LN Pruning (Structured) We calculate weight importance according to an L-n norm for a natural number n in a structured manner [7].

L1 Pruning Global (Unstructured) The threshold according to the L1 norm is applied in the scope of the whole model and not just a single layer. The least important weights of the complete network are pruned [21].

3.4 Quantization

Machine learning model weights tend to use high precision floating point values, usually 32-bit, to represent the weights. While higher precision aims to increase accuracy, running a model is computationally expensive. To reduce both computational and storage expenses, we approximate these weights by representing them with a lower amount of bits, such as FP16 (16-bit), INT8 (8-bit) [50] or as low as a single bit per weight [17]. This reduction of computational load and memory usage allows models deployment on mobile devices or similar hardware limited settings. Without much loss in performance, the simplified computations enable real time results of otherwise long-running models. Another advantage is dealing with rounding errors and other noise introduced by precise floating point values, which makes quantization a great generalization technique.

Quantization is applicable both during (Quantization aware training - QAT) and after the training process (Post training quantization - PQT). QAT includes quantization operations while training a model, allowing it to adapt and optimize its weights for quantized inference. QAT requires a longer training process, but usually yields better results than PTQ and allows fine-tuning the quantization aware training process for specific hardware settings. PTQ, while having slightly drops in accuracy, is simpler and applicable on any pretrained model. Since our interest

is out-of-the-box model compression, our focus is post training INT8 and FP16 quantization [26].

3.5 Hardware Support

Pruning and quantization need different hardware setups to work properly, depending on the specific kind of compression that is in use. Both techniques work on CPUs without many hardships. To accelerate inference, machine learning models usually run on GPUs as they are optimizing matrix multiplications. Pruning changes our weight matrices by either removing weights completely or filling the corresponding values with zero, leading to deformed layers for which we need a GPU with sparse matrix calculation support.

Quantization is changing the usual model precision of FP32 to FP16, INT8 or even lower. While facing no problem on a CPU, most off-the-shelf GPUs do not support integer calculations. Specially designed graphic cards, e.g. NVIDIA's A100, are required to make use of quantization.

TensorRT TensorRT [39] is a machine learning framework by NVIDIA. It optimizes models and inference runtime for execution on (particularly) NVIDIA GPUs. TensorRT focuses on low-latency and high-throughput inference. The library allows the quantization of models from 32-bit floating point (FP32) to 16-bit floating point (FP16) or 8-bit integer (INT8) precision with the help of the TensorRT engine, a runtime engine for network inference. According to NVIDIA, building the engine eliminates dead computations, folds constants and reorders and combines operations for a more efficient run on the GPU with the option of reducing the precision to FP16 and INT8. TensorRT benchmarks layers and computes an optimal execution schedule, minimizing the combined cost of kernel executions and format transforms. [38]. TensorRT is our main tool for quantizing the models to benchmark.

ONNX Open Neural Network Exchange (ONNX) is a machine learning framework built for interoperability of different ML platforms. It is designed to allow the representation, exchange and conversion of models. We use this tool to export our models from PyTorch and calibrate them for and transfer them to TensorRT.

4 Approach

In this section, we cover our line of action for benchmarking model compression techniques. In Subsection 4.1 we discuss our setup of the framework, PQ Bench[1], for evaluating the benchmark results. Subsection 4.2 and Subsection 4.3 give an introduction to the pruners and quantizers we use and which libraries correlate to them.

The goal of this thesis is to gain an overview of the different and most common pruning and quantization techniques used in practice. There are several other approaches on reducing a model's complexity, e.g. dimensionality reduction, knowledge distillation or regularization. But, the key aspect of this thesis is both pruning and quantization. For the techniques to be comparable to each other, we narrow our view to a single field of machine learning, image processing, within the spectrum of computer vision. We focus on those algorithms that are applicable to a given model as out-of-the-box as possible. We join techniques with different already pretrained models and fitting datasets to perform benchmarking tests and evaluate the trade-off between model complexity and performance.

4.1 Setting Overview

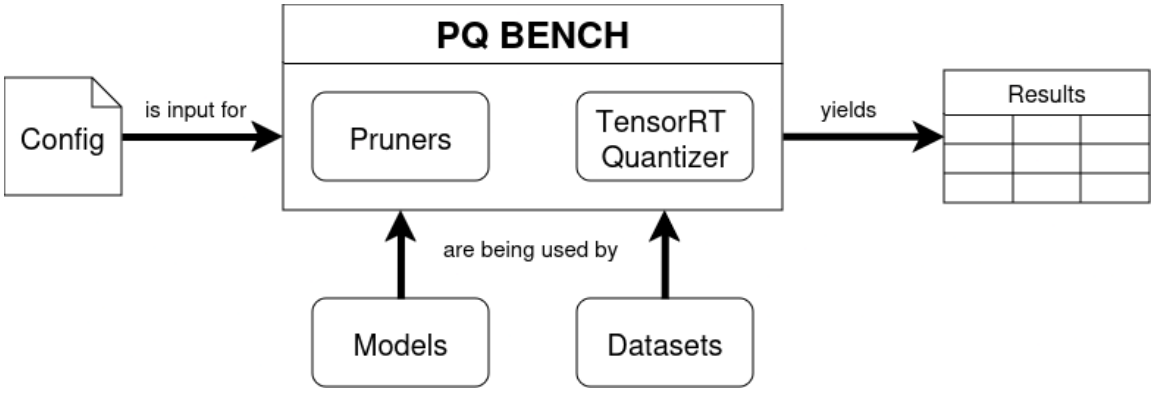


Figure 4: The basic setup of our tool, PQ Bench. The main component for controlling experiment runs is the config file. It defines which subsets of the models, datasets, pruners and quantizers to use.

In Figure 4 we gain an overview of our framework. The main tool is PQ Bench, which runs any combinations and permutations of models, datasets, pruners and quantizing modes. For coherent usage, we implement a Model and Pruner object

[1]The code to this thesis can be found here: `https://github.com/sjoze/master-thesis/tree/main`

and use PyTorch's Dataset object. The Model object simply returns a pretrained model. In our case, we focus on pretrained PyTorch models the TorchHub collection offers. We use PyTorch's DataLoader during execution. A Dataset object therefore implements __init__, __len__, and __getitem__, as required for transformation to a DataLoader. The dataset needs to fit to the pretrained model. We use subsets of ImageNet, as each TorchHub model is pretrained on ImageNet. A Pruner object has the method prune(model), applicable to an input model and prunes according to its initialization, e.g. the pruning degree. Quantization is not an object in itself, as we always use TensorRT for its execution. The only parameter is the choice of INT8 or FP16 quantization. If the user wishes to only use TensorRT's optimization algorithms without quantizing, they have the option on a base run on its engine, summing quantization modes to three in total for our use cases.

To handle benchmarking of all combinations of models, datasets and compression techniques, we use a config file. We set a list of batch sizes to work through and a number of iterations, which determines how many times all configuration combinations run. We measure in three iterations and evaluate the reported median. The config file lists all implemented models and datasets which we enable or disable depending on the experiment. We choose implemented pruners similarly, with an extra list of desired pruning degrees used on every enabled pruner. Including zero runs a model in its base form without any pruning. The structured pruner has parameters for its dimension and Ln norm to use. Quantization has the modes INT8, FP16 and BASE. The config file offers to set the path for storing engine files of TensorRT and experiments. Based on what has been enabled within the config file, every combination runs as many times as set in the iteration setting. PQ Bench iterates through each configuration of a benchmark run, applies the specified compression technique and measures the results by evaluating on the test dataset, all within the tool itself. PQ Bench only collects quantization results with the help of TensorRT, as a quantized model does not run within the PyTorch framework but with TensorRT's inference session. The method of approach to accuracy and inference time measurements is identical in both frameworks.

Table 1: Example CSV results file

Model	Pruner	Quantizer	Dataset	Amount	Accuracy
GoogLeNet	L1 Unstructured	-	ImageNette	0.1	39.10
MobileNet	-	fp16	ImageWoof	-	76.51
AlexNet	L1 Unstr. Global	int8	ImageNette	0.9	92.36

Each run saves to a CSV file with the configuration used and the yielded results i.e. accuracy, batch inference speed and (compressed) model size. In Table 1 we examine how a results file looks like. *Model*, *Pruner*, *Quantizer* and *Dataset* contain

Table 2: Example CSV results file (cont'd)

Inf. Time (ms)	Original Size	Compressed Size	Batch Size	Prep Time (s))
21.624	26.68 MB	26.68 MB	64	0.96
3.836	14.27 MB	7.3 MB	16	55.34
1.846	244.41 MB	62.66 MB	8	18.01

which item has been used. If one of the compression techniques are disabled as a whole, we denote it with a "-". *Amount* is referring to the pruning degree used in pruners. *Accuracy* reports the test accuracy of the compressed model. *Inf. Time (ms)* contains the average time in milliseconds of a batch inference. We only report GPU inference time with PyTorch's profiler and exclude CPU working time so i.e. data loading does not influence inference times. *Original Size* and *Compressed Size* store the measured size of the base model and compressed model, respectively. Non-quantized models measure their pt file size, quantized models measure their TensorRT engine size. *Batch Size* has the batch size of the run. *Prep Time (s)* reports the time in seconds needed for compressing the models i.e. pruning the model, calibrating quantization with ONNX and TensorRT or both.

Besides starting predefined benchmarks, it is possible to initiate a custom run with a self defined model, dataset and pruner / quantizer[2].

4.2 Pruners

We use the pruners included in the PyTorch package [47]. This includes pruning both in a local and global scope for unstructured pruning and pruning with Ln norm for structured pruning. The norm is changeable to an Ln norm for an n of our choosing. The pruner also features a dimension parameter. The most reasonable options are zero, in which case we prune neurons and channels, and one, in which case we disconnect the connections to the inputs.

The pruners of PyTorch fill the pruned weights with zero values. A second approach is provided by the TorchPruner package [33]. Its pruning excludes the weights completely from structurally pruned models. It offers a range of attributions to choose from as criterion for pruning. These include random pruning, weight norm pruning [30], sensitivity pruning [35] and Taylor pruning [36].

4.3 Quantizer

For quantizing, we use NVIDIA's TensorRT [39]. There are different possible workflows to convert a PyTorch model to TensorRT [51]. We choose the quantization with

[2]Further instructions can be found here: `https://github.com/sjoze/master-thesis/blob/main/custom_example.py`

ONNX Runtime TensorRT execution provider [27, 40]. Running an experiment takes the PyTorch model and converts it to ONNX's format. ONNX calibrates the model by computing quantization parameters on a small set of training or evaluation data and exports the resulting model and a calibration file. ONNX execution provider uses hardware acceleration libraries and computes optimized execution subgraphs during this step [51]. TensorRT now configures a TensorrtExecutionProvider, which receives a full precision ONNX model to use, the calibration settings and whether to apply FP16 and INT8 quantization. We start an inference session that calculates how to apply the quantization logic and executes our model with the given quantization calibration. The results, like during pruning, save to the current experiment CSV.

5 Evaluation

In this section, we compare the different model compression approaches and their influence on performance, inference speed and model size. In Subsection 5.1 and 5.2 we cover the models and datasets used for benchmarking. In Subsection 5.3 we investigate the change in accuracy, in Subsection 5.4 potential speed-up during inference and in Subsection 5.5 the impact on the size of a compressed model. In Subsection 5.6 we take a look at the potential of combining pruning and quantization, and in Subsection 5.7 the influence of compression techniques on a model architecture other than CNNs, transformers.

Hardware We use an NVIDIA V100 with 32GB GPU for our benchmarks. This choice supports the usage of NVIDIA's TensorRT quantization framework. We run on Python 3.11.5, PyTorch 2.4.1, TensorRT 10.4.0, CUDA 12.2 and ONNX 1.16.2 with ONNX Runtime GPU support 1.19.2. The reported values represent the median of three runs. The variance of the reported values is small enough that we skip visualizing the confidence interval for the sake of clarity.

We use a warm-up of ten iterations of batch inference before each benchmark. This ensures that our GPU is not in idle mode and gives realistic results. We evaluate all models featured in Subsection 5.1, but mostly report on AlexNet and MobileNet as representatives for loose, unoptimized and compressed, optimized models respectively. We are using a batch size of 128, if not state otherwise.

5.1 Models

We cover a variety of models with focus on CNNs. We use pretrained models from TorchHub that range from early CNNs like AlexNet to modern, compact models for limited hardware settings on edge devices like MobileNet. To maintain the aspect of minimizing overhead, we remain with their collection.

Table 3: Model Specs

Model	#Parameters	Size (MB)	Reference
AlexNet	61,100,840	244.41	[28]
ResNet (18)	11,689,512	46.83	[19]
DenseNet (121)	7,978,856	32.48	[20]
GoogLeNet	6,624,904	26.66	[45]
Inception (V3)	27,161,264	108.96	[46]
MobileNet (V2)	3,504,872	14.25	[42]
SqueezeNet	1,248,424	5.01	[23]
Shuffle	2,278,604	9.28	[32]
Simple	5,752,808	23.07	[18]

AlexNet AlexNet (2012) was, for its time, a deep and highly effective CNN designed for GPUs consisting of five convolutional layers followed by three fully connected layers.

ResNet Residual Neural Networks (2015) are CNNs which implement a residual block that allows weights to skip layers. The amount of convolutional layers range from 18 to 152.

DenseNet DenseNet (2017) layers receive input not just from the direct preceding layer but from multiple layers prior with concatenation. This has proven to be computationally efficient and preserve low complexity.

GoogLeNet Using a neural network module called Inception, GoogLeNet (2015) utilizes this architecture to choose between different convolutional filter sizes. This allows the net to grow deeper without a huge spike in amount of parameters.

Inception Inception V3 (2016) is the third iteration of the inception family, based upon GoogLeNet, improved by adding factorized convolutions and regularization in the form of label smoothing.

MobileNet MobileNet (2017), MobileNetV2 (2018) and MobileNetV3 (2019) are convolutional neural networks with a hardware-aware architecture designed for mobile phones.

SqueezeNet SqueezeNet (2016) is designed as a smaller neural network that 'squeezes' parameters to reduce their amount while still achieving competitive accuracy.

ShuffleNet ShuffleNet (2017), similar to MobileNet, is designed for mobile devices which gets its name by implementing a technique to shuffle channels in convolutions.

SimpleNet SimpleNet (2016) is prioritizing a straightforward architecture with mere 13 layers.

5.2 Datasets

We choose subsets of ImageNet as our datasets because the models of Subsection 5.1 are pretrained on the ImageNet collection. We focus on ImageNette and ImageWoof with its ten classes each as they vary in difficulty of classification.

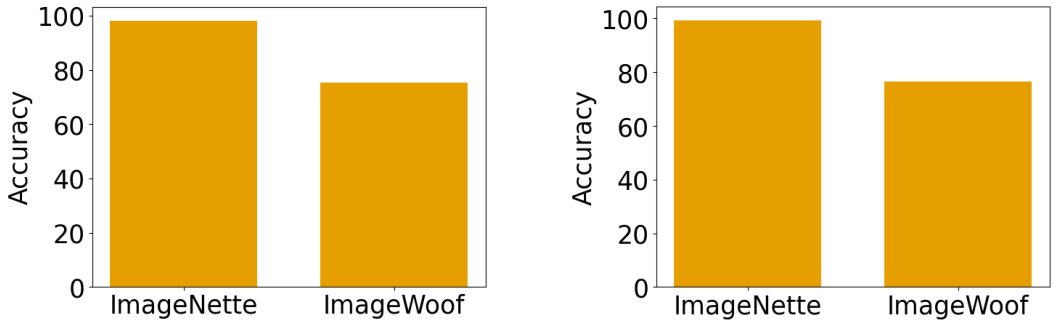
ImageNet ImageNet contains over 14 million manually annotated training images, widely used in image classification and object detection benchmarks. It covers roughly 20,000 classes, with each over 500 images on average.

ImageNette ImageNette is a subset of ImageNet, containing only ten easily classified classes of sizes 320 px and 160 px.

ImageWoof ImageWoof is a subset of ImageNet, containing ten dog breeds which tend to be harder to classify than the more distinguishable objects of ImageNette.

5.3 Accuracy

In this subsection, we analyze the effect of pruning and quantization on model accuracy. Accuracy is a common measure for the performance of a model. Pruning and quantization aim to preserve accuracy as best as possible while gaining advantages in terms of speed and model size. We investigate to what degree different techniques maintain their levels of accuracy while removing parameters.

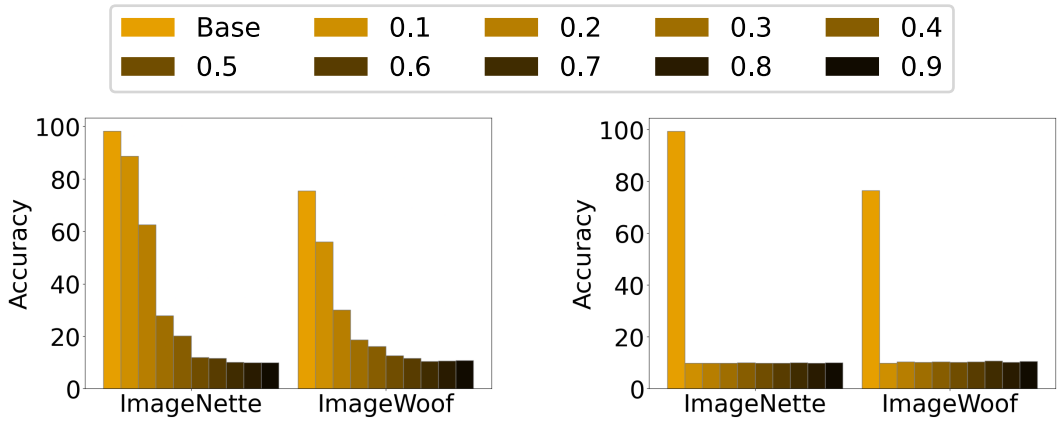


(a) AlexNet (b) MobileNet

Figure 5: Comparing accuracy on base AlexNet and MobileNet.

In Figure 5 we see how the pretrained base models from PyTorch's collection perform on our test dataset split. AlexNet achieves an accuracy of 98.29% and MobileNet a near perfect 99.41% on the easier to classify ImageNette. On ImageWoof the models still reach a mark of 75.48% and 76.51% respectively. MobileNet is outperforming AlexNet as the more modern model.

Pruning (Zero Fill) The first compression technique is pruning while filling removed values with zero entries. This retains the integral structure of the base model and is easy to apply since we do not have to take caution in rescaling input and output layers. The only operation needed is modifying the weights' values.



(a) AlexNet (b) MobileNet

Figure 6: AlexNet and MobileNet pruned using random pruning.

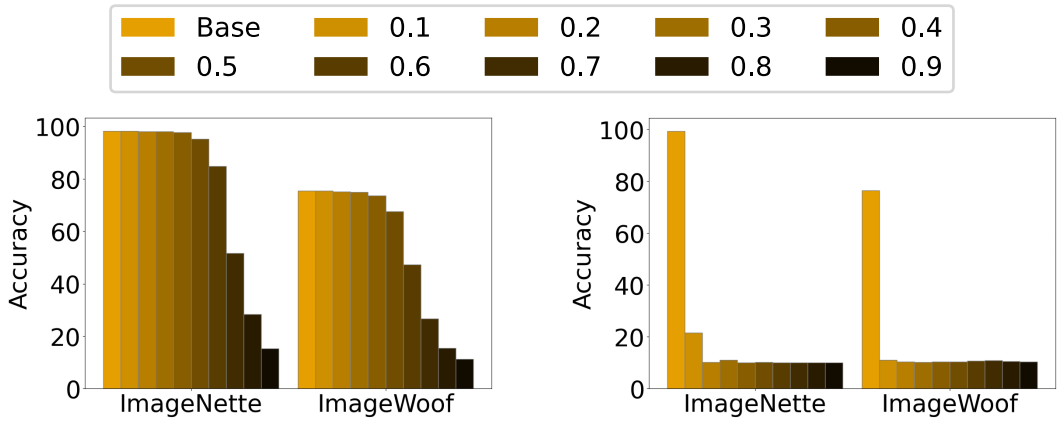
In Figure 6 we examine the performance drop-off in regard to accuracy on randomly pruned AlexNet on MobileNet models with increasing pruning degrees. We increase the amount of pruned weights by ten percent on each experiment. A ten percent pruned model already has ten percent decreased accuracy on ImageNette and twenty percent decreased accuracy on ImageWoof. If we prune twenty percent of weights, the model achieves slightly better than half of its base accuracy on ImageNette. Accuracy performance flattens at the bottom with large amount of pruning. AlexNet is guessing classes (ten percent accuracy for ten classes to classify) at a level of fifty percent pruned and beyond, making the model unusable.

MobileNet is, due to it being designed for edge devices, both optimized and compact. The slightest degree of random pruning causes the model to fully lose its functionality and be on par with randomly guessing the label.

This behavior is expected as AlexNet is less compact and rather large, and thus more open to model compression and the usual choice for papers covering pruning. MobileNet is more optimized and slimmer, and the accuracy is taking an even more severe hit. Random pruning heavily decreases accuracy in both cases and renders both models essentially unusable, especially in the case of models such as MobileNet.

In Figure 7 we analyze the effect of using the unstructured L1 norm as criterion for choosing the weights to prune. Compared to random pruning, AlexNet has significantly improved performance. Up to a pruning degree of fifty percent, we notice little to no difference in accuracy. Even with only a tenth of the weights remaining, the model is borderline unusable, but still performs slightly better than randomly guessing (averages around thirteen percent accuracy).

The L1 norm weight removal finds little success when we apply it to an optimized model such as MobileNet. The model drops off just slightly less harsh than when randomly pruning, but still severe enough that even a ten percent pruned MobileNet



(a) AlexNet (b) MobileNet

Figure 7: AlexNet and MobileNet pruned using L1 unstructured pruning.

has an accuracy below thirty percent.

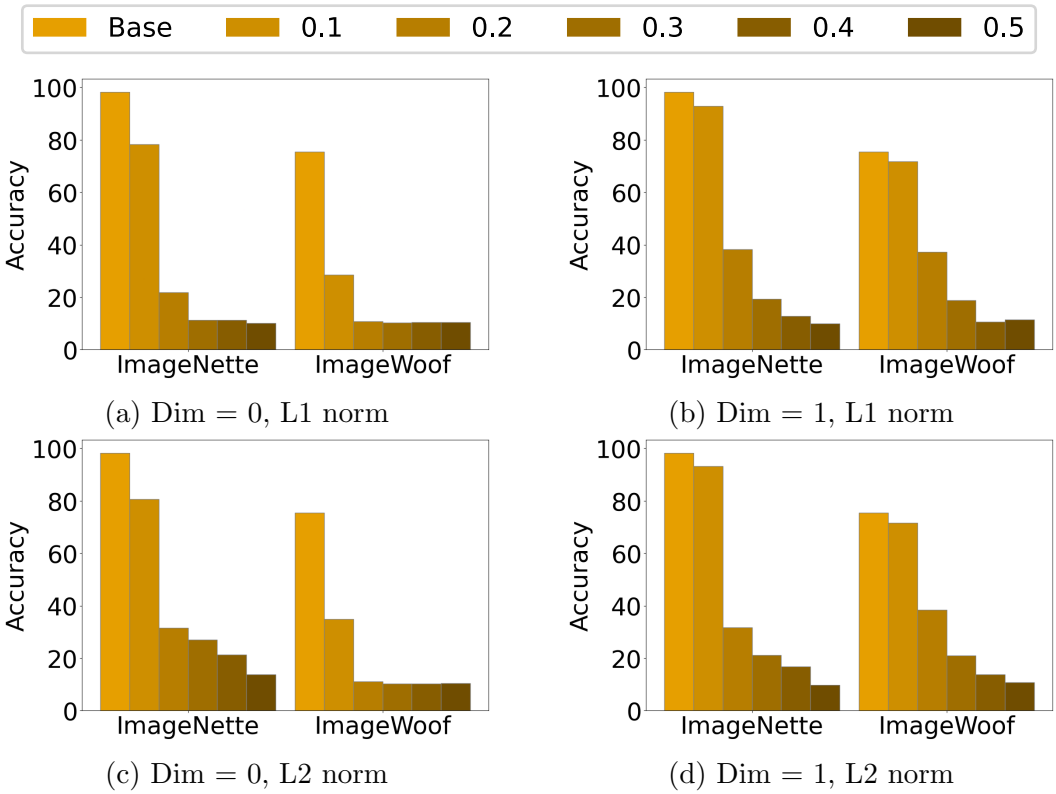
As expected, not randomly guessing which weights to prune but taking their influence into account with the L1 norm improves pruning results. AlexNet benefits more from the change in pruning criterion than MobileNet.

In structured pruning, we apply the same norm rule but disconnect channels and filters as a whole to retain the integrity of the model as opposed to pruning isolated weights.

In Figure 8 we see the impact of all permutations using the L1 and L2 norm respectively. We visualize up to a pruning degree of fifty percent, as accuracy flattens out to random guessing beyond that. The results look similar to what we see in unstructured random pruning. Accuracy drops off fast in each of the four combinations of configurations. We examine that the norm does not change results in a great sense. A slightly higher improvement is the pruning of neuron input connections rather than neurons themselves. But, any of these techniques do not outperform the unstructured L1 pruning.

We assume that structured pruning, while maintaining structural integrity of the model, does not have the freedom of precisely choosing individual weights with little importance to remove. Structured pruning limits itself to removing e.g. whole filters, in which a majority of weights might be unimportant but still feature weights of significance which also get removed in the process. Unstructured pruning pinpoints low impact weights more accurately.

In Figure 9 we analyze the effect of pruning a model on a global scale. We apply L1 unstructured pruning in steps of ten percent pruning degree over the whole neural network. The change in scope shows close to no drop-off in accuracy for AlexNet. With ten percent of weights left in the model, it still performs well. We



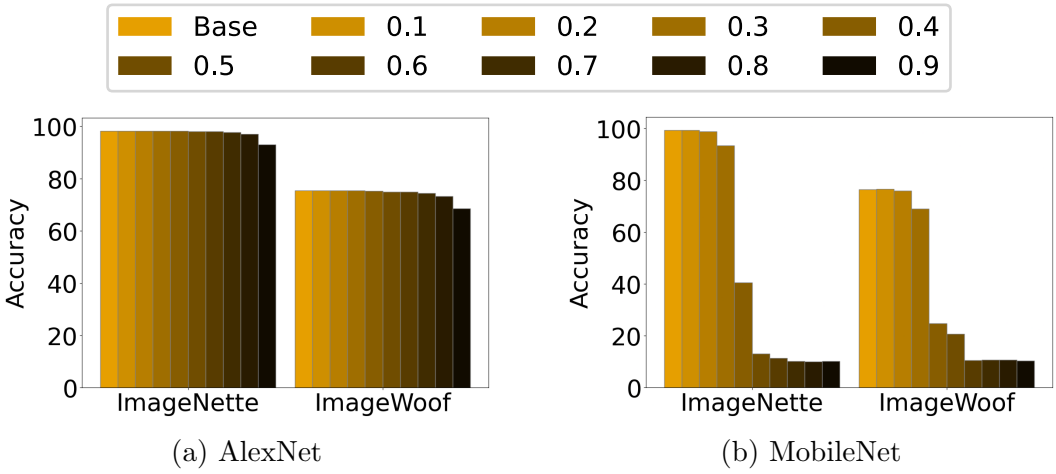
(a) Dim = 0, L1 norm

(b) Dim = 1, L1 norm

(c) Dim = 0, L2 norm

(d) Dim = 1, L2 norm

Figure 8: AlexNet pruned using structured pruning on different dimensions and norms.



(a) AlexNet

(b) MobileNet

Figure 9: AlexNet and MobileNet pruned using L1 global unstructured pruning.

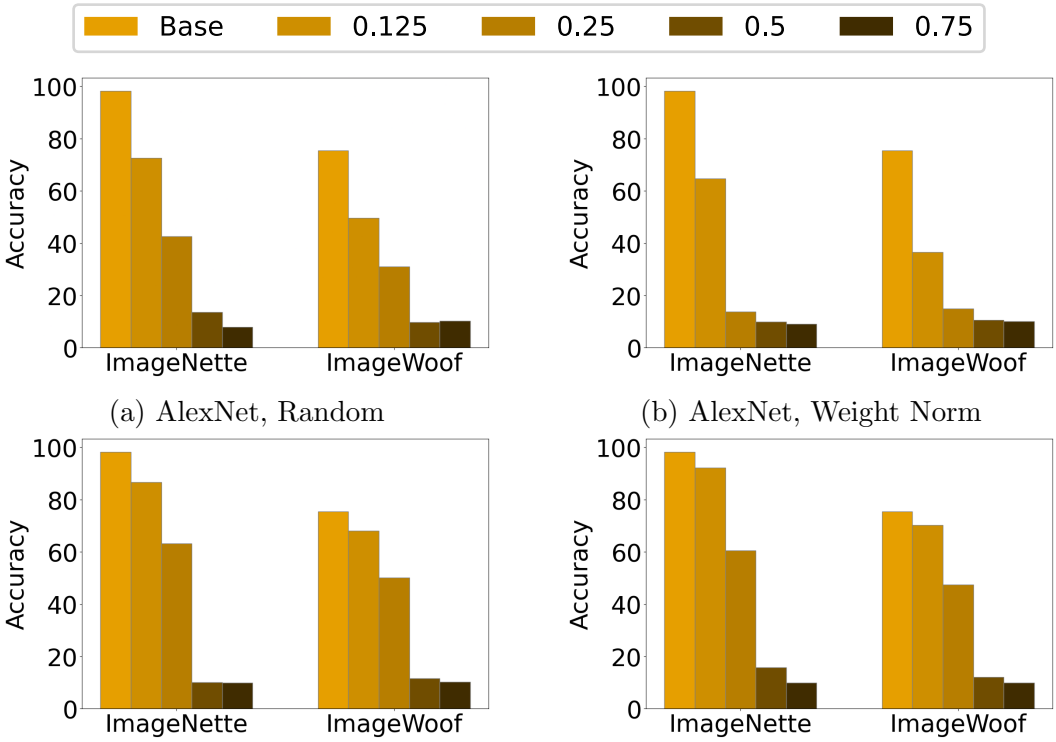
do not lose any performance in a compact and presumable optimized model such MobileNet with twenty percent of weights pruned. Up to a third of the weights

pruned, MobileNet maintains competitive performance.

Our explanation is that pruning in a global scale evaluates which layers are important to the inference and retain their weights. Taking the whole model into account, we nullify more irrelevant weights and are not restrained to keeping weights of layers with little impact or forced to remove weights of influential layers.

Pruning (Removal) So far, we prune weights by setting their value to zero. Pruning with removal physically removes those weights from the model and thus changes the shape of its layers. This works smoothly for merely AlexNet and SimpleNet. Once models become less linear and more complex, i.e. residual blocks and skip connections are in use, we have trouble calculating how to prune input and output shapes accordingly and make them match each other.

We run structured pruning with all four criteria, random, weight norm, sensitivity and Taylor in degrees of multiples of one eighth to account for shapely layer sizes of powers of two.



(a) AlexNet, Random

(b) AlexNet, Weight Norm

(c) AlexNet, Sensitivity

(d) AlexNet, Taylor

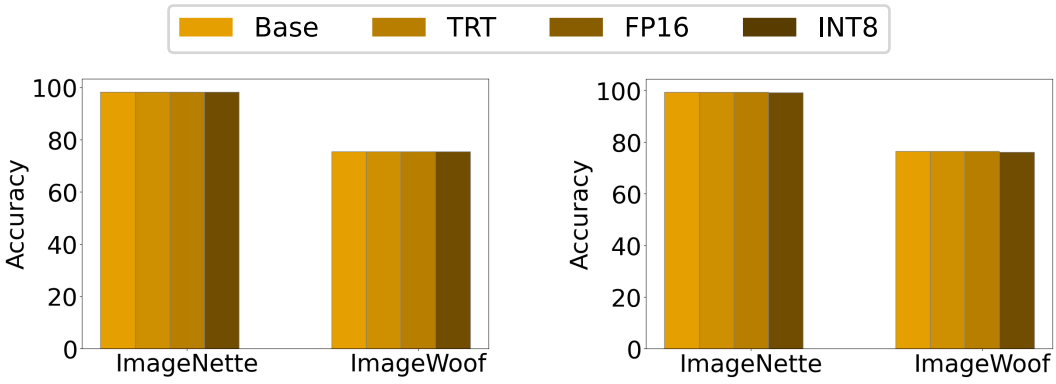
Figure 10: Comparing attributions of pruning with weight removal on AlexNet.

In Figure 10 we examine the impact of the different techniques on accuracy. Accuracy drops off fast with every attribution. Random pruning slightly outperforms

the weight norm. Out of the four options, Taylor pruning, which evaluates the importance of each weight using a second-order approximation (Taylor expansion), is the best with the least amount of drop-off in accuracy.

We see that weight removal with structured pruning, similar to the zero filled case, reduces accuracy more heavy than the unstructured pendant. Our explanation is the same as with zero filled pruning, as a structured approach is more limited in its choice for poor performing weights as opposed to unstructured pruning.

Quantization Quantization introduces model compression by reducing the precision of weights from FP32 to FP16 or INT8. NVIDIA's TensorRT engine allows performing inference on quantized models on GPUs with FP16 or INT8 support. Our expectation is that calculations of weights in a base model are unnecessarily precise and offer similar results when quantized.



(a) AlexNet (b) MobileNet

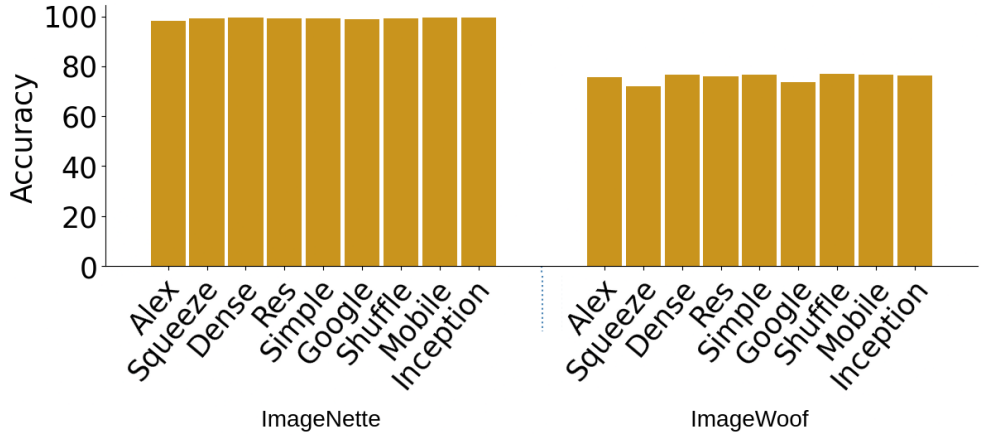
Figure 11: Comparing quantization on AlexNet and MobileNet.

In Figure 11 we examine the accuracy of quantized AlexNets and MobileNets. The results reveal virtually no drop-off in accuracy on both AlexNet and a compact model such as MobileNet.

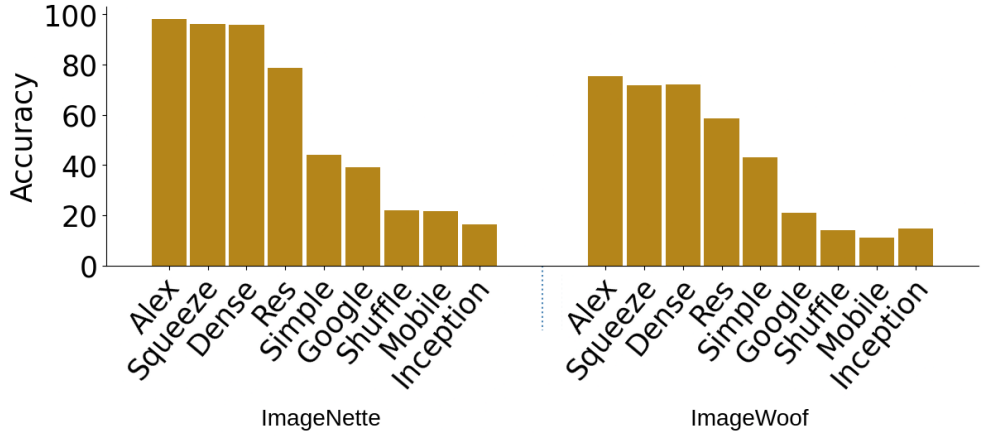
Our expectations are met that FP32 introduces too much precision and a quarter of the designated bit depth is sufficient for the same accuracy performance.

In summary, the best results, with emphasis on accuracy, come from quantizing a pretrained model or pruning it on a global scale. The latter only allows unstructured criteria, but we have seen that these outperform structured pruning anyway. Quantization yields overall better results than pruning. We expect this is because it preserves the structure of the model rather than removing weights entirely, which retains the model's ability to represent complex patterns. The reduction of the precision of weights introduces some approximation errors, but these are less significant than the impact of pruning large portions of the network. On a model as compact as

MobileNet, quantizing allows us to use one fourth of the base precision and pruning one third of the base weights without losing a noteworthy amount of accuracy.

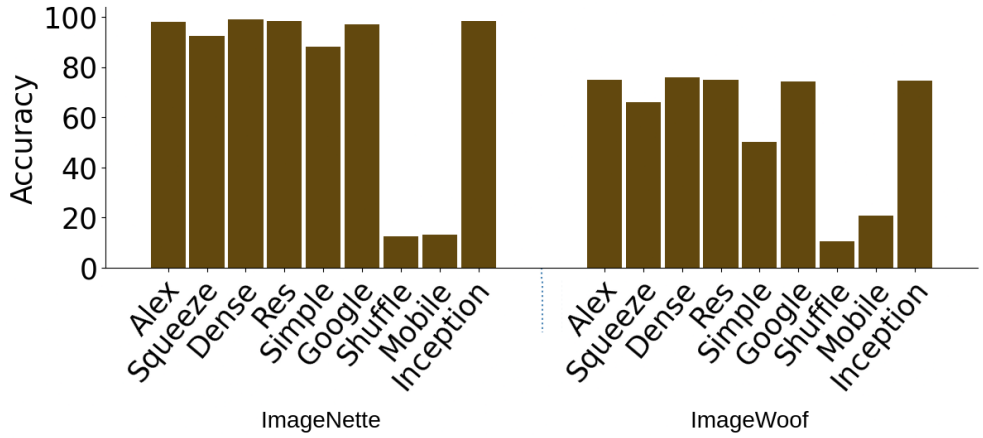


(a) Accuracy of different unpruned models.

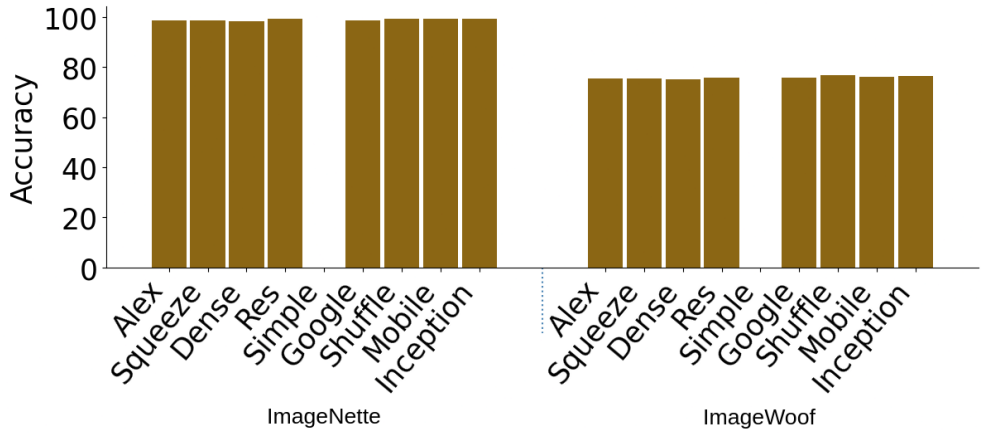


(b) Accuracy of locally slightly pruned models (degree = 0.1).

In Figure 12 we gain an overview of common pretrained models which all score relatively similar accuracy values when not compressed. When pruning ten percent of weights, we examine what has been indicated in the examples of AlexNet and MobileNet. Unoptimized models such as Dense- and ResNet are candidates for slight compression without heavy accuracy loss, whilst compact models as Inception and GoogleNet essentially lose their functionality upon pruning. But if we apply the same L1 norm pruning strategy on a global scale, most models retain their base accuracy even with half of the weights pruned. Only ShuffleNet and MobileNet succumb under the removal of half their weights. Every other model still scores an accuracy of eighty percent or above on ImageNette. A noticeable outlier is the Inception model, which has the least amount of accuracy when locally pruning a tenth of weights, being one of the better scoring models under global pruning, contrary to expectations. Quantization has even better results than global pruning.



(c) Accuracy of globally moderately pruned models (degree = 0.5).



(d) Accuracy of INT8 quantized models.

Figure 12: Comparing pruning and quantization impact on different models. Pruning using the L1 norm, both in a local and global scope. Quantization in INT8 precision.

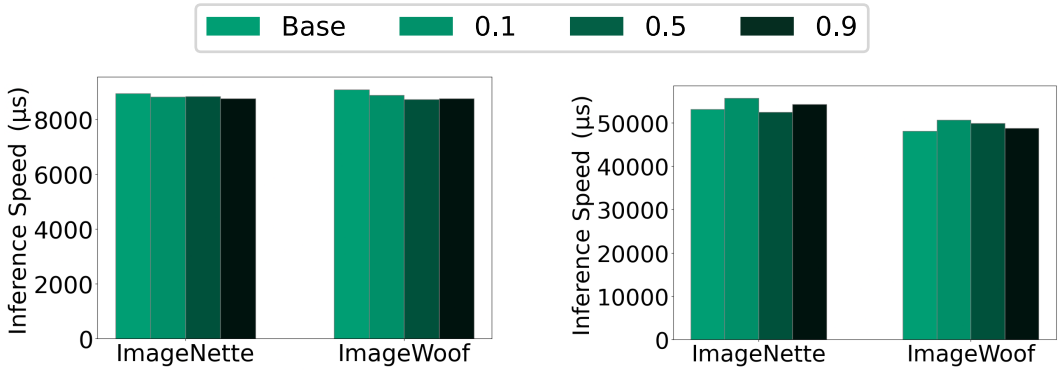
With a quarter of the base precision, every model, without exception, performs as well as its respective base comparison[3].

5.4 Inference Speed

In this subsection, we analyze the impact of compression on inference speed. A faster model means faster decision-making, which leads to a better user experience and potentially real time processing. Faster models benefit scalability as they handle more requests and lower latency. The reduced load also enables usage in hardware limited settings and on edge devices. We examine the influence of pruning and

[3]SimpleNet was unable to be quantized

quantization on the time to process a test batch on average.



(a) AlexNet (b) MobileNet

Figure 13: Comparing zero fill pruning impact on inference speed with local unstructured L1 pruning.

Pruning (Zero Fill) In Figure 13 we examine the speed of models with zero filled values for pruned weights. Pruning takes less than a few seconds to apply, but the speed-up gained is negligible, as a pruned model has virtually no increase in pace. There is close to no difference in speed between a lightly and heavily pruned model. This behavior is prevalent in all our tested models.

Our explanation is that a pruned model with zero entries does not completely ignore these weights on a forward pass. In theory, we do not need to multiply zero entry weights, as their result are always zero. During inference, the neural network takes advantage of matrix multiplication, on which GPUs are specialized on. But, these multiplications always take the whole matrix into account. And since we replace non-zero entries with zero rather than actually reducing the size or dimensions of the weight matrix, we do not gain a significant speed-up.

Pruning (Removal) When removing pruned weights from our model, we physically shrink the model and skip any matrix calculations involving pruned entries. In Figure 14 we see the inference speed of a structurally pruned AlexNet with removed weights. Again, we include steps with coverage of every eight of a pruning degree for the sake of keeping models shapely. We examine, compared to zero fill pruning, an actual change in inference speed. Models pruned to a degree divisible by one quarter tend to be faster than resized models with other layer shapes. In general, the time for a batch inference steadily declines as expected, with a slight plateau at the end. NVIDIA's TensorRT promises support for sparse matrix inference on selected GPUs, but they demand significant preprocessing steps and only work on exactly fifty percent sparse models [25].

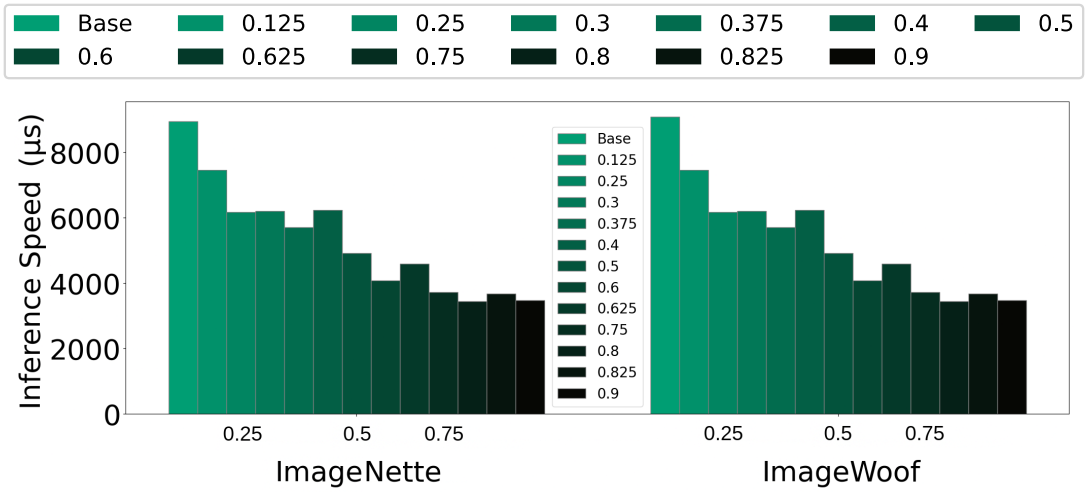
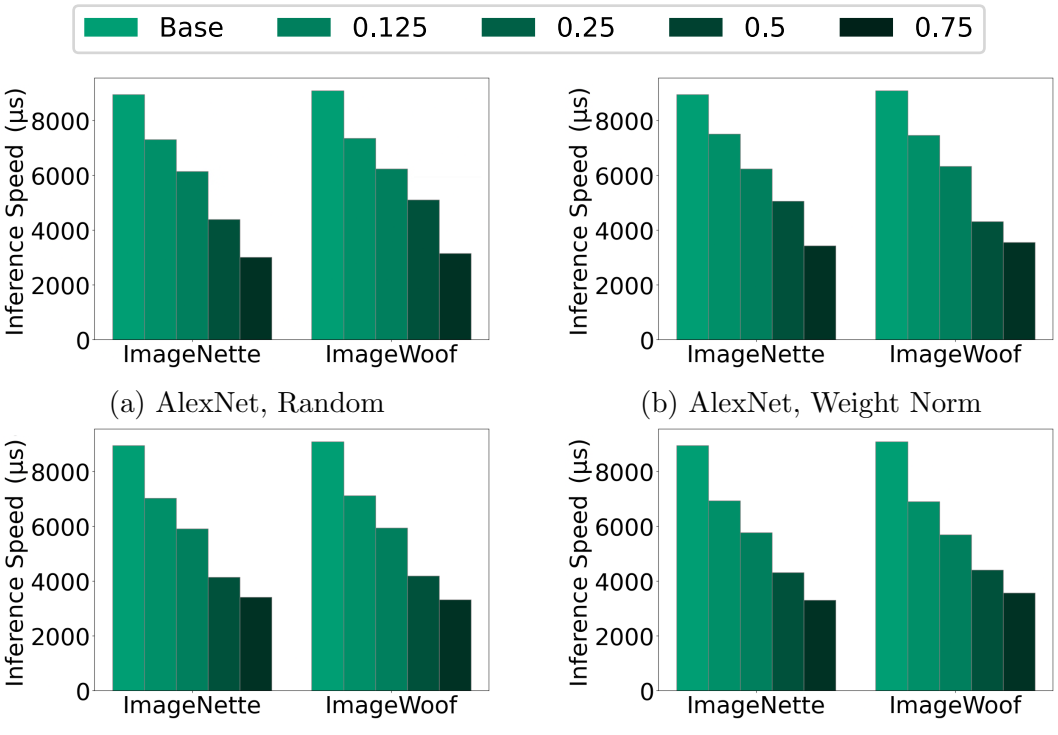


Figure 14: Batch inference speed of an L1 structured pruned AlexNet with weight removal.



(a) AlexNet, Random

(b) AlexNet, Weight Norm

(c) AlexNet, Sensitivity

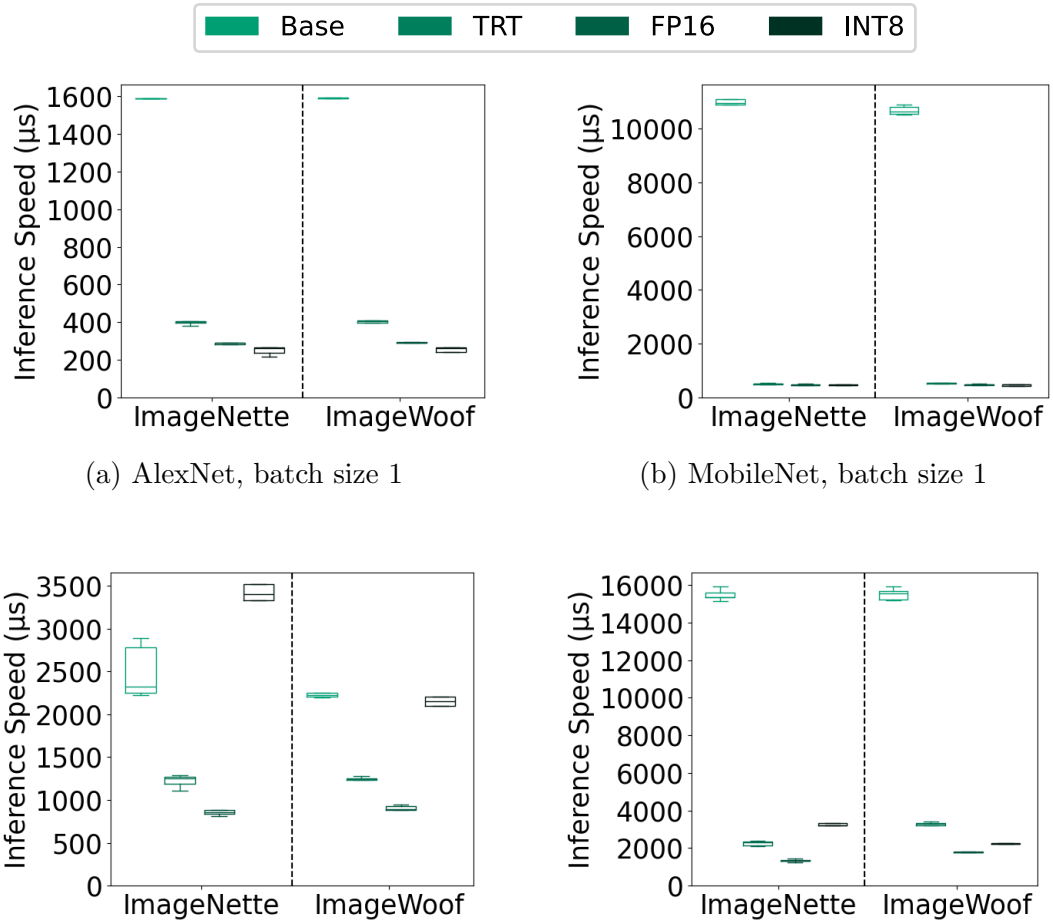
(d) AlexNet, Taylor

Figure 15: Comparing attributions of pruning with weight removal on AlexNet.

In Figure 15 we examine the inference speed when pruning with our different attributions, random, weight, sensitivity and Taylor pruning. Each seem to have

the same impact on the reduction of batch processing time. The resulting model size is the only significant factor in our speed-up, and not the criterion determining it, as is to be expected.

Quantization Quantization decreases the precision of our model's weights, which simplifies calculations during inference. We expect, with equal support for FP32, FP16 and INT8, a rough speed-up of twice the inference time from FP32 to FP16 and FP16 to INT8, respectively.



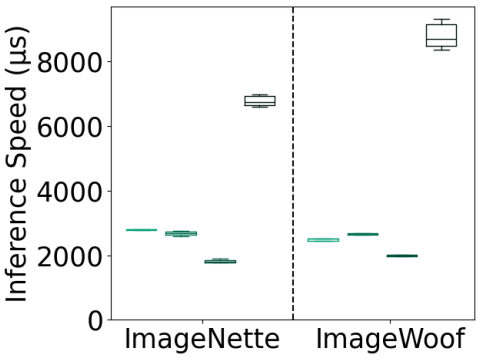
(a) AlexNet, batch size 1

(b) MobileNet, batch size 1

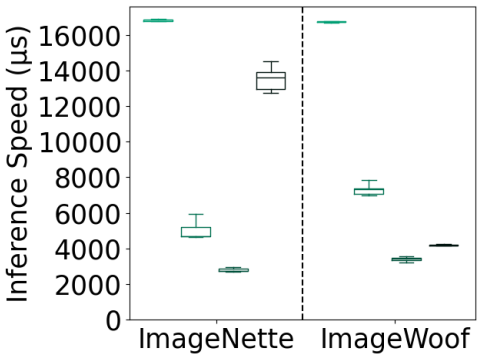
(c) AlexNet, batch size 8

(d) MobileNet, batch size 8

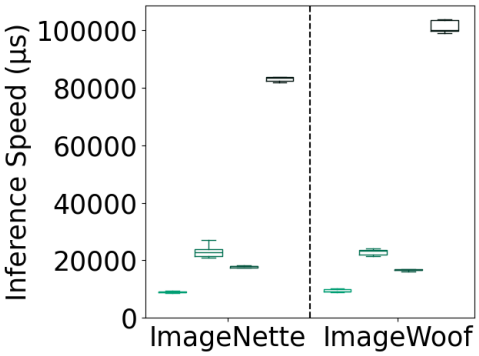
In Figure 16 we see inference speed results on AlexNet and MobileNet quantized in TensorRT with different batch sizes. Due to comparatively higher variance in reported inference speed, we choose to box plot the results of five runs instead of the usual three. With a batch size of one, we get faster with each reduction of precision, whilst the largest gain in speed is due to the usage of the TensorRT engine in contrast to a raw PyTorch run. With a batch size of eight, we see that running an INT8 quantized model significantly increases the time to process the



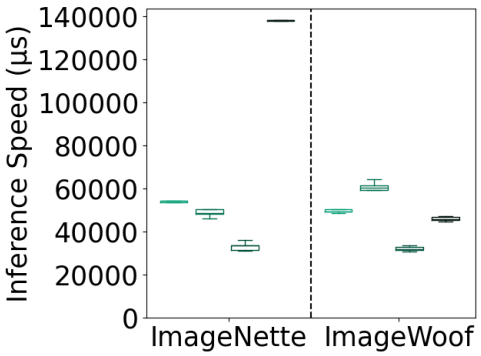
(e) AlexNet, batch size 16



(f) MobileNet, batch size 16



(g) AlexNet, batch size 128



(h) MobileNet, batch size 128

Figure 16: Comparing quantization impact on inference speed with AlexNet and MobileNet with different batch sizes

batch. In AlexNet it surpasses FP16 and the basic TensorRT engine run and infers slower than the PyTorch execution on ImageNette. With increasing batch size the slowdown of INT8 quantization becomes greater whilst TensorRT in general shows signs of missing scalability in regard to batch size. At a batch size of 128, a lot of PyTorch model executions are faster than TensorRT, with or without quantization. The preparation and calibration for quantization takes less than two minutes, in most cases less than one minute. For low batch sizes, the investment pays off after several inference runs.

Quantization meets our expectations only in a single item inference setting. Increasing the batch size correlates with poorer performance of the whole TensorRT engine. Especially INT8 quantization yields disappointing results. Our assumption lies on the tensor core of a V100. A tensor core is a specialized hardware unit in NVIDIA GPUs designed to accelerate tensor (a matrix with many dimensions) computations. Although our GPU has native INT8 and FP16, the V100's chip's tensor core only supports FP16. In cases where INT8 quantization is not applicable,

TensorRT falls back on FP16 quantization.

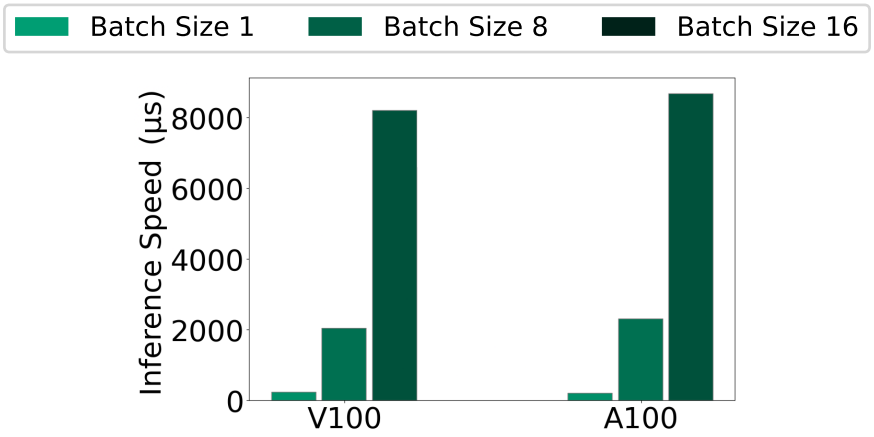


Figure 17: AlexNet on ImageNette, INT8 quantized on a V100 and A100.

We expected the bad results regarding the INT8 inference times due to the usage of a V100. We repeat INT8 quantizing on an NVIDIA A100-SXM4-80GB. INT8 tensor cores are available for Xavier, Turing and Ampere GPUs. In Figure 17 we examine INT8 quantization results on different batch sizes. We see similar reported values, hinting that even stronger GPUs still do not fully support integer quantization.

5.5 Model Size

In this subsection, we investigate the effect of compression techniques on model size. A more compact model with competitive performance offers several advantages. A smaller model with fewer parameters is, on paper, faster than a bigger model. It benefits storage and execution in hardware limited settings and reduces power consumption. We analyze if pruning or quantizing weights actually translates to more lightweight models.

Table 4: Model size of zero fill pruned AlexNet

Model	Size (MB)	Size (MB), 20% pruned	Size (MB), 90% pruned
AlexNet	244.41	244.41	244.41

Pruning (Zero Fill) Zero fill pruning causes the removal of weights by replacing them with a zero entry. Naturally, the model size remains the same, as the structure and reserved bytes of our model are identical. As seen in Table 4, no matter how

much the neural network has been pruned, its size matches the base model's. This behavior is prevalent throughout all models tested.

By default, PyTorch even doubles the size of the model. This is because the original weights are being kept unchanged, whilst the result of the pruning process is saved as a mask of equal size. To apply the changes, we need to explicitly enforce the mask to regain the same model size as prior to pruning.

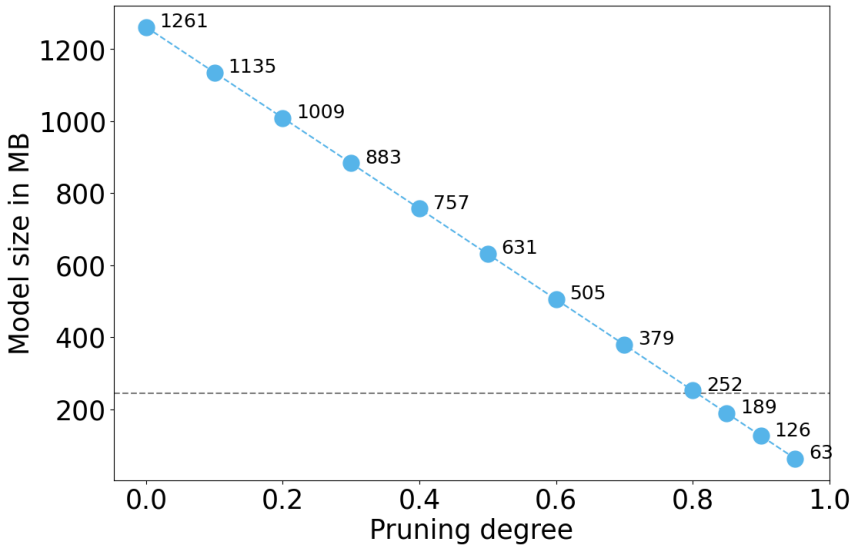


Figure 18: AlexNet pruned and stored using a sparse matrix representation to tackle the growing majority of zero value entries. Even at eighty percent sparsity, a pruned sparse model is still larger than the original model.

A model with a high frequency of zero entries in its weights stands to reason that a sparse matrix representation decreases a model size, at least during storage. PyTorch implements sparsity through coordinate format. As examined in Figure 18, the model size linearly decreases, but the threshold for a sparse model to actually become smaller than the base model (here: 244.41 MB), is beyond 80 percent sparsity. A model such as AlexNet retains a decent amount of performance, though most other modern compact models' accuracies have already lost a significant amount of accuracy at this point.

Pruning (Removal) Contrary to pruning with filling zero values, when physically removing the weights, we expect to see a shrinkage in model size. In Figure 19 we examine a logarithmic decrease in model size. At thirty percent of weights removed, AlexNet is half as big as compared to the base model. The overall reduction meets our expectations.

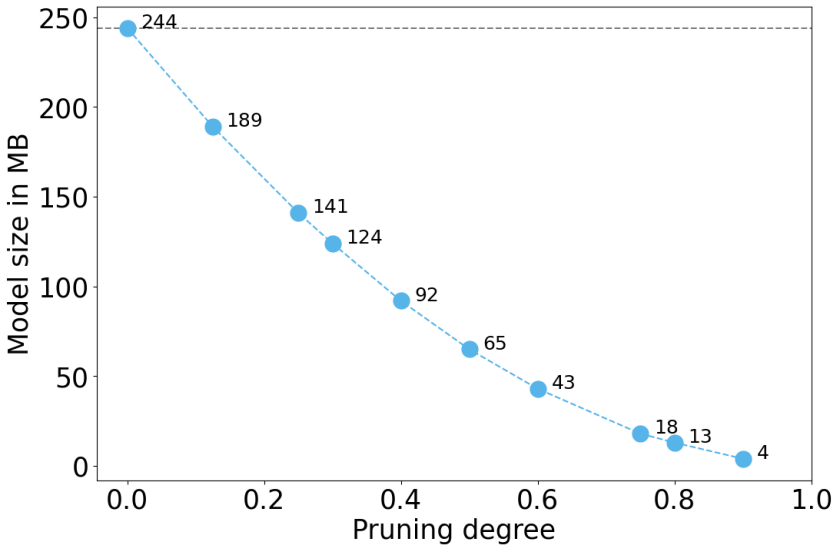
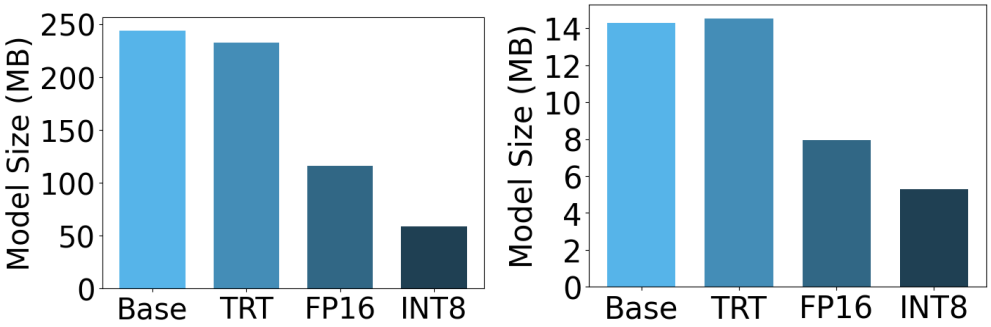


Figure 19: AlexNet pruned using structured pruning and removing the weights from the layers.

Quantization As quantization happens inside TensorRT, we do not regain a typical PyTorch model after preprocessing but an executable TensorRT engine, which is a runtime engine for network inference. The serialized form of the engine forms the execution plan and is stored to disk. The following values represent the size of our engine file compared to our base model.



(a) AlexNet (b) MobileNet

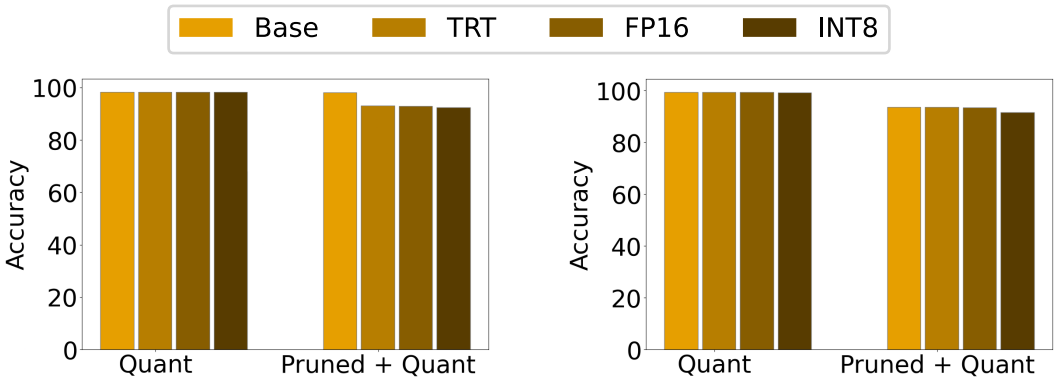
Figure 20: Comparing quantization impact on model size.

In Figure 20 we examine the model size of TensorRT engines compared to AlexNet's and MobileNet's base model. The non-quantized engine is roughly as big as the underlying PyTorch model. FP16 quantization halves the model size compared to FP32, as expected. In the case of AlexNet and MobileNet, the models meet our expectation by being also half as big when INT8 quantized. But, all other models share only the shrinkage during FP16 quantization. INT8 quantized models

are as large as the FP16 pendant. To our best knowledge, we do not examine an explainable pattern but assume that some model architectures are not supported by TensorRT's integer quantization.

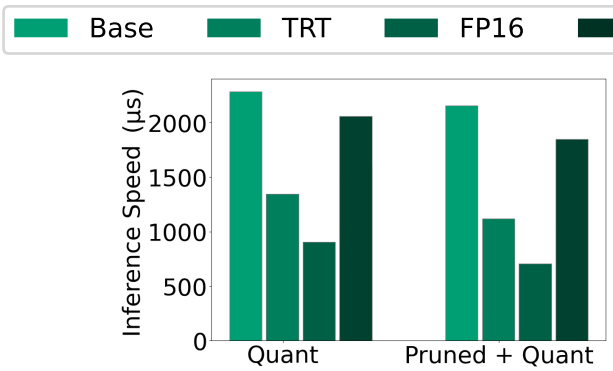
5.6 Pruning in Combination with Quantization

Pruning and quantization are combinable by pruning weights of a base model and then quantizing the remaining. In the previous sections, we see that global unstructured pruning yields the best ratio of pruned weights to maintained accuracy. Structured pruning with removal shrinks model size, but loses too much performance while doing so. We investigate combining our best pruner, global pruning, with INT8 quantization and expect slightly worse but still competitive accuracy results while probably being en par with the inference speed of just quantizing.



(a) Accuracy of a quantized and a 90% global pruned + quantized AlexNet.

(b) Accuracy of a quantized and a 30% global pruned + quantized MobileNet.



(c) Inference speed of a quantized and a 90% global pruned + quantized AlexNet, batch size 8.

Figure 21: Analyzing the combination of pruning and quantization on AlexNet and MobileNet.

In Figure 21 we examine the combination of a ninety percent globally unstructured pruned AlexNet with quantization on ImageNette. A pruned but not quantized AlexNet scores 93.06 percent accuracy. A quantized but not pruned AlexNet scores 98.30 percent. Both compression techniques combined still hold on to 92.36 percent accuracy. MobileNet, when pruned to a degree of thirty percent and INT8 quantized, still retains 91.62 percent of its base 99.41 percent accuracy on ImageNette. This exceeds our expectations, as a presumably optimized and compressed MobileNet still scores above ninety percent accuracy while having one third of its weights missing and only one quarter of its base precision.

We see the inference speed of a quantized AlexNet compared to a pruned and quantized AlexNet. Batch inference times are reduced by ten to fifteen percent on average when pruned prior. A zero fill pruned model is not smaller than the base model and without sparse matrix calculation support, we do not take full advantage of the abundance of zero entries. We imagine GPUs with sparsity support to accelerate the model even greater.

But, as we keep the INT8 engine's reduced 'model size' of a quarter compared to the original FP32 size, we get the most out of model size and inference time reduction for the least amount of accuracy loss with this approach.

5.7 Pruning and Quantization on Transformers

In this Subsection, we take a quick look at Vision Transformers and the influence of pruning and quantization on a different architecture than CNNs. Vision Transformers play a big role in the field of image analysis and deep learning. Raising their efficiency through model compression improves a state-of-the-art technique such as the transformer architecture even further.

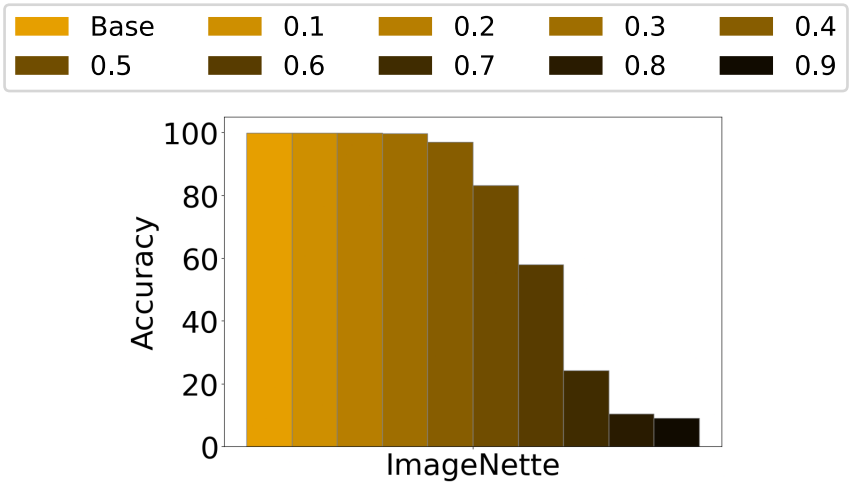


Figure 22: Vision Transformer pruned globally unstructured on ImageNette.

Pruning In Figure 22 we examine the impact of global unstructured pruning on a pretrained Vit_b_16 model [13] from TorchVision. We see accuracy en par with the base model up to a degree of thirty percent. Accuracy drops below ninety percent only after removing half of the weights.

The success on a transformer architecture is better than we expected. Half of the Vision Transformers's weights are not impactful to the model performance, and the model is reducible to half its size in theory.

Quantization Quantizing the Vision Transformer model does not impact accuracy at all. The base, FP16 and INT8 all score the same 99.92 percent on ImageNette. FP16 and INT8 are both twice as fast (8 ms to 4 ms batch inference compared to a TensorRT run without quantization) and half in engine size (346 MB to 170 MB compared to the base model). This implies that while quantization also finds success on transformers, there is no difference between FP16 and INT8 quantization. We assume this is because even high-end NVIDIA GPUs do not fully support integer operations on any given architecture type.

6 Summary and Discussion

In this section, we build upon the findings of Section 5 and conclude our results. We cover further potential work on this topic in Section 7.

The goal of this master thesis was to investigate two of the most common model compression techniques, pruning and quantization. We used tools as out-of-the-box from the PyTorch library as possible to analyze the convenience and effect of available compression strategies on given models. We saw that pruning, as expected, has greater effect on an unoptimized model rather than a compact one. Unstructured pruning with zero values resulted in better accuracy values, but had no advantage in terms of inference speed. Structured pruning with removal of weights yielded smaller, faster models but does not hold up with accuracy performance. Global, unstructured pruning gave the best results in contrast to local pruning. The expansion of scope to the whole model benefits accuracy retention. A support of sparse matrix computation during inference would make this technique very promising.

Quantization with TensorRT has shown little performance loss while speeding up inference by a great amount on small batch sizes, admittedly most by optimization steps of TensorRT itself. FP16 quantization outperformed INT8 quantization due to absence of support for integer optimization. A GPU with full INT8 quantization and sparse matrix inference support implies to improve a both pruned and quantized model remarkably.

6.1 The Value of Pruning

The most promising form of post training pruning according to our results is pruning globally. We are able to prune a large amount of weights of unoptimized models without any loss in accuracy. But, global pruning is exclusive to unstructured pruning. This makes it difficult to remove weights and gain a benefit in model size. A speed-up in inference is possible if we run on a GPU that fully implements and supports sparse matrix calculations. In theory, we accelerate inference greatly without much loss in performance. But, without sparsity support, zero filled models are not significantly faster than base models. We only achieve accelerated models with weight removal pruning, in which case accuracy loss correlates heavily with inference speed gained.

6.2 Feasibility of Quantization

Quantization is supported with PyTorch packages [48] and allows both post training quantization and quantization aware training. But, these libraries support running quantized models on CPU but not on GPU. To benefit to the full extent with the inference speed gained, a user is forced to use TensorRT which is exclusive to high-end NVIDIA GPUs and common middle class hardware finds no use. As TensorRT

is an early prototype, INT8 quantization is not fully implemented and does not benefit from the same optimizations as FP16.

6.3 Pruning vs Quantization

Quantization with TensorRT combines both a small drop-off in accuracy and a high gain in inference speed. Pruning theoretically matches or outperforms quantization in case of sparse matrix inference support. It would be interesting to investigate a pruned and quantized model on hardware that fully implements INT8 quantization and sparse matrix calculations, as this promises great results according to what our findings hint at.

7 Future Work

Different Model Architectures So far we limited ourselves to investigating convolutional neural networks pretrained on ImageNet. However, image analysis machine learning models include more architectures than CNNs, as for example transformers. We could also expand the scope even further and include neural networks besides CNNs. The impact of compression techniques on different architectures and the comparison to one another is an interesting topic.

Quantization on commonware GPUs We have seen that quantization, especially integer quantization, is limited to special high-end GPUs that support these operations. As the results of quantization seem promising, we could work on facilitating compressing models on off-the-shelf GPUs.

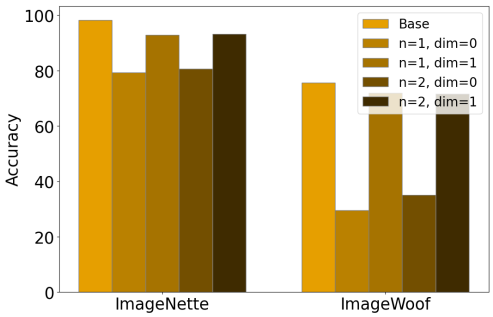
Quantization below INT8 Quantization is possible down to a single bit level. As we have seen virtually no drop-off in accuracy on even INT8 quantized models, it is interesting to investigate to what degree of quantization each model type is possible.

Pruning with removal on all model types Weight removal pruning has been limited to simple, straightforward model architectures. It would be very interesting to expand the pruning package to allow structured pruning on more complex model architectures that feature, i.e. residual- and transformer blocks.

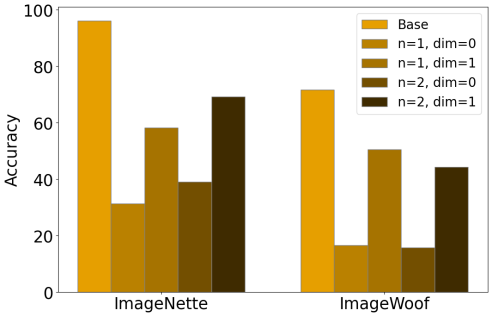
Recommendation of compression techniques We have benchmarked pruning and quantization techniques. But future work could build on top of that by expanding the benchmarking tool to a recommendation tool, potentially including specified hardware specs to determine which compression technique is most suitable for the user.

More metrics We have covered accuracy, inference speed and the size of the model so far. Future work could benchmark e.g. economically valuable metrics such as power usage (and, in that sense, savings through model compression).

8 Appendix

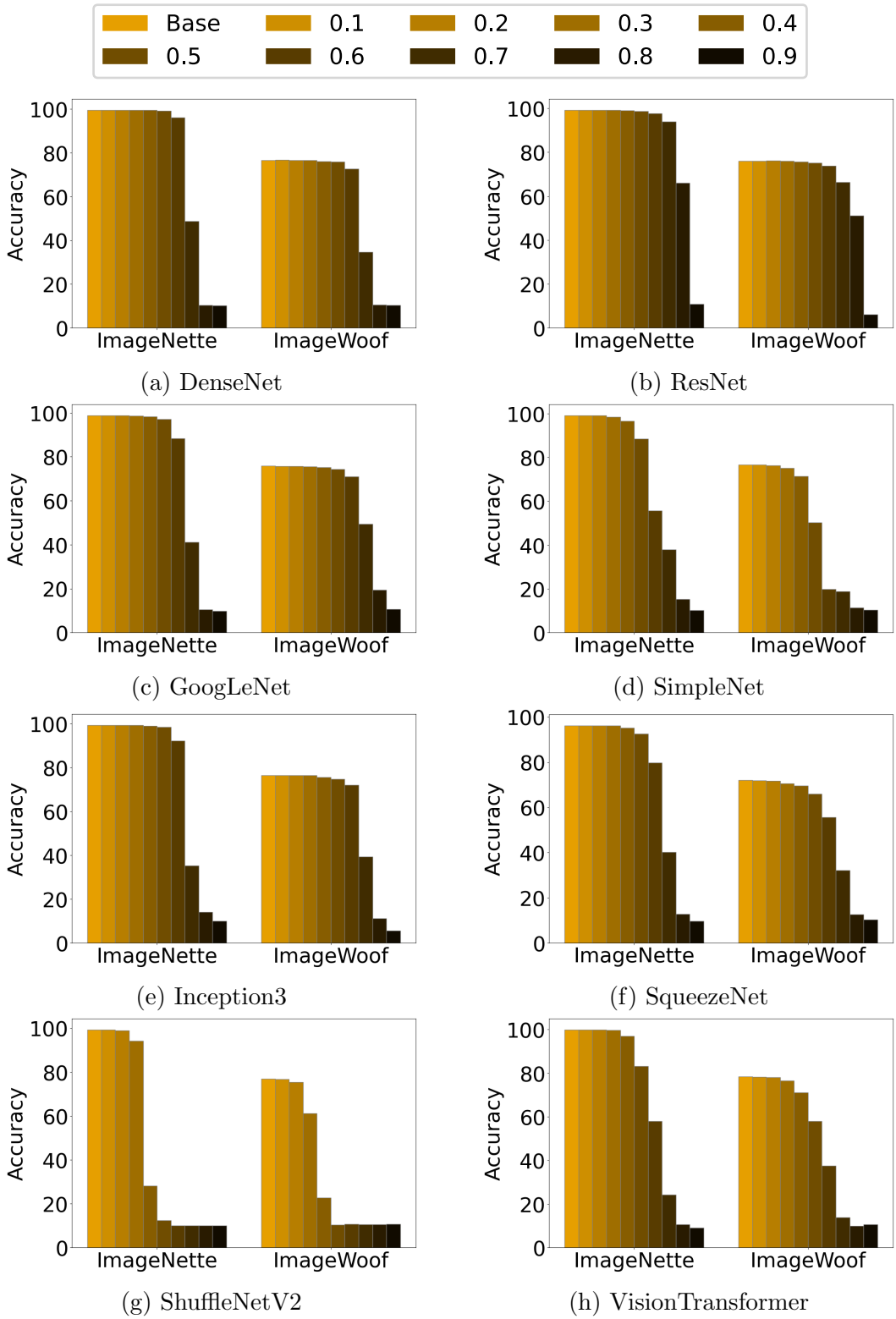


(a) Accuracy of AlexNet with structured pruning.



(b) Accuracy of SqueezeNet with structured pruning.

Figure 23: AlexNet and SqueezeNet structurally pruned (degree = 0.1), different norms (1 and 2) and different dimensions (0 and 1), batch size 8.



(a) DenseNet (b) ResNet

(c) GoogLeNet (d) SimpleNet

(e) Inception3 (f) SqueezeNet

(g) ShuffleNetV2 (h) VisionTransformer

Figure 24: All other models globally unstructured pruned.

References

[1] Amazon. Was ist der Unterschied zwischen KI und Machine Learning? https://aws.amazon.com/de/compare/the-difference-between-artificial-intelligence-and-machine-learning/. Accessed: 2024-10-15.

[2] M Augasta and Thangairulappan Kathirvalavakumar. Pruning algorithms of neural networks—a comparative study. *Open Computer Science*, 3(3):105–115, 2013.

[3] M Gethsiyal Augasta and T Kathirvalavakumar. A novel pruning algorithm for optimizing feedforward neural network of classification problems. *Neural processing letters*, 34:241–258, 2011.

[4] Ummara Bibi, Mahrukh Mazhar, Dilshad Sabir, Muhammad Fasih Uddin Butt, Ali Hassan, Mustansar Ali Ghazanfar, Arshad Ali Khan, and Wadood Abdul. Advances in pruning and quantization for natural language processing. *IEEE Access*, 12:139113–139128, 2024.

[5] 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.

[6] Blalock, Davis and Gonzalez Ortiz, Jose Javier and Frankle, Jonathan and Guttag, John. Shrinkbench. https://github.com/JJGO/shrinkbench. Accessed: 2024-11-27.

[7] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[8] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019.

[9] Cloudflare. Wie funktionieren neuronale netzwerke? https://www.cloudflare.com/de-de/learning/ai/what-is-neural-network/. Accessed: 2024-10-15.

[10] Eda Kavlakoglu Cole Stryker. What is ai? https://www.ibm.com/topics/artificial-intelligence. Updated: 2024-08-16, Accessed: 2024-10-15.

[11] DeepAI. Artificial intelligence. https://deepai.org/machine-learning-glossary-and-terms/artificial-intelligence. Accessed: 2024-10-15.

[12] DeepAI. Deep learning. https://deepai.org/machine-learning-glossary-and-terms/deep-learning. Accessed: 2024-10-15.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[14] Andries P Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE transactions on Neural Networks*, 12(6):1386–1399, 2001.

[15] Eirik Fladmark, Muhammad Hamza Sajjad, and Laura Brinkholm Justesen. Exploring the performance of pruning methods in neural networks: An empirical study of

the lottery ticket hypothesis, 2023.

[16] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[17] RaviKiran Gopalan and Oliver M. Collins. An optimization approach to single-bit quantization. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(12):2655–2668, 2009.

[18] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures, 2023.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[21] Zhongzhan Huang, Wenqi Shao, Xinjiang Wang, Liang Lin, and Ping Luo. Rethinking the pruning criteria for convolutional neural network. *Advances in Neural Information Processing Systems*, 34:16305–16318, 2021.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018.

[23] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

[24] Java Point. Pruning in Machine Learning. `https://www.javatpoint.com/pruning-in-machine-learning`. Accessed: 2024-10-18.

[25] Jeff Pool, Abhishek Sawarkar, Jay Rodge. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT. `https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/`. Accessed: 2024-11-25.

[26] Joel Nicholls. Quantization in Deep Learning. `https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b`. Accessed: 2024-10-18.

[27] Katsuya Hyodo. TensorRT Excecution Provide. `https://zenn.dev/pinto0309/scraps/42587e1074fc53`. Accessed: 2024-11-27.

[28] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks, 2014.

[29] Andrey Kuzmin, Markus Nagel, Mart Van Baalen, Arash Behboodi, and Tijmen Blankevoort. Pruning vs quantization: Which is better? *Advances in Neural Information Processing Systems*, 36, 2024.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, 2017.

[31] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[32] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical

guidelines for efficient cnn architecture design, 2018.
[33] Marco Ancona. TorchPruner. https://github.com/marcoancona/TorchPruner/tree/master. Accessed: 2024-11-12.
[34] Mark Kurtz. What is Pruning in Machine Learning? https://opendatascience.com/what-is-pruning-in-machine-learning/. Accessed: 2024-10-18.
[35] Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, and Balaraman Ravindran. Studying the plasticity in deep convolutional neural networks using random pruning, 2018.
[36] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.
[37] Prateeth Nayak, David Zhang, and Sek Chai. Bit efficient quantization for deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 52–56. IEEE, 2019.
[38] NVIDIA. NVIDIA TensorRT Documentation. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html. Accessed: 2024-11-25.
[39] NVIDIA. TensorRT. https://developer.nvidia.com/tensorrt. Accessed: 2024-11-17.
[40] ONNX. ONNX Execution Provider. https://onnxruntime.ai/docs/execution-providers/TensorRT-ExecutionProvider.html. Accessed: 2024-11-24.
[41] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. A comprehensive survey on model quantization for deep neural networks. *arXiv preprint arXiv:2205.07877*, 2022.
[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.
[43] Suryabhan Singh, Kirti Sharma, Brijesh Kumar Karna, and Pethuru Raj. Pruning and quantization for deeper artificial intelligence (ai) model optimization. In Sanjay Sharma, Bidyadhar Subudhi, and Umesh Kumar Sahu, editors, *Intelligent Control, Robotics, and Industrial Automation*, pages 933–945, Singapore, 2023. Springer Nature Singapore.
[44] Stanford University. Convolutional Neural Networks (CNNs / ConvNets). https://cs231n.github.io/convolutional-networks/. Accessed: 2024-10-15.
[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.
[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.
[47] Torch. Torch Prune. https://pytorch.org/docs/stable/nn.html. Accessed: 2024-11-19.
[48] Torch. Torch Quant. https://pytorch.org/docs/main/quantization.html. Accessed: 2024-11-19.
[49] Sunil Vadera and Salem Ameen. Methods for pruning deep neural networks. *IEEE Access*, 10:63280–63300, 2022.
[50] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. In-

teger quantization for deep learning inference: Principles and empirical evaluation, 2020.

[51] Yuxiao Zhou, Zhishan Guo, Zheng Dong, and Kecheng Yang. Tensorrt implementations of model quantization on edge soc. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 486–493. IEEE, 2023.

[52] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.