DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE

EasyCompress

Automated Compression for Deep Learning Models

THESIS

Abel Van Steenweghen

4876431

Contents

ABSTRACT

Over the past years the size of deep learning models has been growing consistently. This growth has led to significant improvements in performance, but at the expense of increased computational resource demands. Compression techniques can be used to improve the efficiency of deep learning models by shrinking their size and computational needs, while preserving performance.

This thesis presents EasyCompress, an automated and user-friendly tool to compress deep learning models. The tool improves on existing compression research by focusing on generalizability and practical usability, in three ways. Firstly, it aligns with specific compression objectives and performance requirements, ensuring the compression accomplishes its intended goal effectively. Secondly, it employs flexible compression techniques, so that it is applicable to a diverse set of models without requiring deep model knowledge. Finally, it automates the compression process, eliminating difficult and time-consuming implementation efforts.

EasyCompress intelligently selects, tailors, and combines various compression techniques to minimize model size, latency, or number of computations while preserving performance. It employs structured pruning to reduce the number of parameters and computations, uses knowledge distillation techniques to ensure better accuracy recovery, and uses quantization to achieve additional compression.

The tool's effectiveness is evaluated across diverse model architectures and configurations. Experimental results on a range of models and datasets demonstrate its ability to reduce the model size at least 5-fold, inference time by at least 1.5-fold, and the number of computations by at least 3-fold. Most compression rates are even higher, reaching up to 10, 20, and even 100-fold reductions.

The tool is available online at `https://thesis.abelvansteenweghen.com`.

1 INTRODUCTION

Deep learning has become a cornerstone of artificial intelligence, enabling machines to learn from data and make predictions or decisions in a wide range of applications. It has become the most popular field of research in machine learning and the majority of advances on the state-of-art in artificial intelligence are coming from this field [4, 43].

However, the success of deep learning comes with a significant computational cost, which is rising [42, 10]. The size of these large deep learning models poses a series of problems. First and foremost, because of the large amounts of memory and processing power required to train and deploy large models, they are impractical for resource-constrained devices such as mobile phones or IoT devices. Second, the need for computational resources makes it practically impossible for smaller teams or individual researchers without the computational resources to work with these large models. Third, large models consume more energy, which is not only financially costly but also contributes to the growing carbon footprint of deep learning models [42].

To address these issues, researchers have created various compression methods, designed to reduce the size and complexity of deep neural networks while preserving their accuracy. These methods aim to eliminate redundant or unnecessary parameters, minimize the precision of model weights, distill knowledge from larger models into smaller ones, or find any other mechanism to reduce the size, inference time and number of computations of a model.

Current deep learning compression techniques have proven to be effective, but they present several challenges. First, the dependency on specialized hardware or software can limit the accessibility and usability of compressed models, making it difficult for a wide range of users and applications to leverage their benefits. Second, the wide variety of model architectures, such as CNNs and Transformers, make it difficult to find the right compression techniques. It is important to leverage different techniques to target the unique characteristics of each architecture. Third, many compression techniques require a deep level of understanding of the model's inner workings to apply them. This makes applying them less flexible, and more time-intensive. Lastly, different models need different balances between compression and performance. Depending on the intended application of the model we might have a different threshold for required performance, need a different degree of compression, or focus on a different type of compression such as model size, energy usage, or inference time reduction.

The aim of this thesis was to build a tool that automates the compression process in a way that prioritizes generalizability and practical usability. This research was driven by and aims to answer the following research questions:

- How can the compression process be automated?
- How can compression techniques be combined in a complementary way?
- How can the user's requirements be used to devise a set of compression techniques that achieves these requirements.
- How does the compression goal influence the selection of the most suitable combination of techniques?
- How well does technique combination preserve accuracy for compression goals?

The primary objectives of this research are:

- Develop a user-friendly tool that automates the process of compressing deep learning models, requiring only the trained model, the dataset it is trained and evaluated on, and the performance requirements of the user.
- Ensure that the compressed model is capable of running on standard hardware and execution environments.
- Investigate how compression techniques can be combined in a complementary manner.
- Evaluate the tool's performance across diverse model architectures and configurations, including CNNs and transformers.

To summarize, this thesis presents a tool for compressing deep learning models in an automated and approachable manner, focusing on the user's compression objective and performance requirements. It is designed to compress models without the need for deep model knowledge and allows the compressed model to be used for inference without a need for specialized hardware or execution environments. It creates a set of compression actions and tailors their configurations based on the model's properties and the specific needs of the user. These compression actions are then implemented using an automated algorithm to deliver a compressed model that meets the compression objective.

2 Background

This section goes over the current landscape of compression techniques for deep learning models. It focuses on the fundamentals of the three main compression categories: pruning, knowledge distillation, and quantization. For each category, the main techniques are discussed, highlighting their underlying principles, strengths, and limitations.

2.1 Pruning

Pruning is a widely adopted model compression technique that aims to remove redundant or less important parameters from deep learning models, thereby reducing their size and computational complexity. The central idea behind pruning is that not all parameters in a deep learning model contribute equally to its performance, and eliminating those with minor contributions can lead to a more compact and efficient model. This approach can be broadly classified into structured and unstructured pruning, which differ based on the granularity of the parameters being removed. In recent years, several pruning algorithms have been proposed, with varying degrees of success in achieving model compression while maintaining performance. This subsection provides an overview of the main pruning approaches, along with their respective strengths, weaknesses, and recent advances.

2.1.1 Unstructured Pruning

Unstructured pruning involves the removal of individual parameters, such as weights or connections, rather than entire structures. Individual weights in the model are removed based on a predetermined threshold or ranking criteria. This approach can lead to higher compression rates compared to structured pruning, as it allows for a finer-grained control over which parameters are removed. However, it may result in irregular data access patterns, which can hamper hardware acceleration.

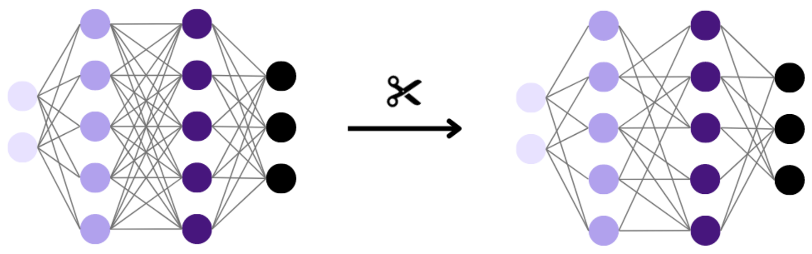

Figure 1: Example of unstructured weight pruning

Unstructured pruning techniques can be differentiated based on how they score parameters. Magnitude-based pruning [21] is the most common unstructured pruning technique. It removes weights with small magnitudes (below a certain threshold) from the network. The threshold can be set based on a percentage of smallest weights, a specific value, or other criteria. Gradient-based pruning [35, 33] prunes weights with small gradients during training. This assumes that weights with smaller gradients contribute less to the learning process and can be removed without significant loss in performance. Hessian-based pruning [32] utilizes the second-order information (Hessian matrix) of the network's loss function to identify and prune less important weights. It is computationally expensive but can yield better results in some cases.

An important concept in unstructured pruning is the Lottery Ticket Hypothesis [15]. This is an iterative pruning technique that starts by training a dense network, then pruning it and resetting the remaining weights to their initial values. The idea is to find a subnetwork (the "winning ticket") that can be trained to the same accuracy as the original dense network but with fewer weights.

While unstructured pruning techniques have demonstrated many promising results, translating the sparsity into actual metric reductions is difficult because of the need for regularity [2]. Regularity refers to the structured organization of weights in a model after applying the pruning method. Regularity is important because it ensures that the pruned network can still be efficiently computed on hardware accelerators, such as GPUs and TPUs, which are optimized for dense matrix computations. Unstructured pruning, which removes individual weights in a model, can result in irregularly sparse weight matrices. With a sparsity-aware inference runtime such as DeepSparse from Neural Magic [24], this can lead to great performance. However, the sparse weight matrices are difficult to accelerate on standard commercial hardware, such as laptops or standard GPUs.

2.1.2 Structured Pruning

Structured pruning refers to the removal of entire structures within the model, such as neurons, filters, or even layers. This approach maintains the regularity of the remaining structures and enables efficient hardware acceleration, as it preserves the original data layout. However, it has a higher chance of pruning important parameters, decreasing the performance.

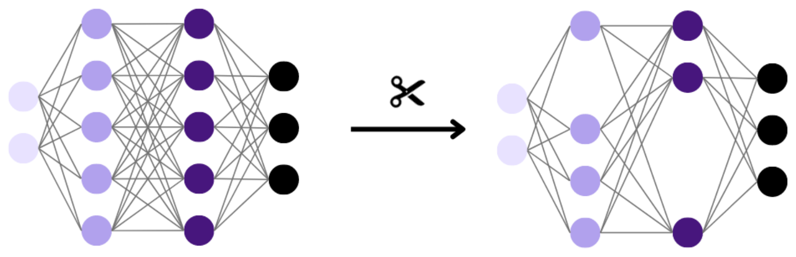

Figure 2: Example of structured neuron pruning

Structured pruning techniques can be differentiated based on the type of structure they prune. The most common structured pruning techniques include filter pruning, block pruning, channel pruning, and layer pruning. Filter pruning targets the removal of filters in convolutional layers, which results in a reduced number of output channels. This approach has been shown to be effective in compressing convolutional neural networks (CNNs) while maintaining their performance. [31] Channel pruning, on the other hand, focuses on removing entire channels of feature maps, leading to a reduced number of input channels for subsequent layers. [30, 36] Layer pruning, although less common, involves the removal of entire layers within the model, which can be beneficial in cases where specific layers contribute little to the overall performance. [38]

2.1.3 Pruning Algorithms

Various algorithms have been proposed for pruning deep learning models, with most of them falling under one of two categories: one-shot pruning [5] and iterative pruning [3, 17]. In one-shot pruning, the model is pruned once, either during or after training, based on a single criterion. This approach is computationally efficient but may lead to suboptimal pruning decisions. Iterative pruning, on the other hand, involves multiple rounds of pruning and fine-tuning, allowing the model to gradually adapt to the removal of parameters. This approach typically yields better results than one-shot pruning, albeit at a higher computational cost.

2.2 Quantization

Quantization is a model compression technique that focuses on reducing the precision of the parameters in a deep learning model, such as weights and activations, by representing them with fewer bits. For example, a weight matrix that is typically represented in a single-precision floating format (FP32) can be mapped to a new weight matrix where the weights are in an 8-bit integer format (INT8), as represented in Figure 3. This example would lead to a 4x reduction in size. These reductions in bit representation lead to a significant decrease in the model's memory footprint and computational requirements.

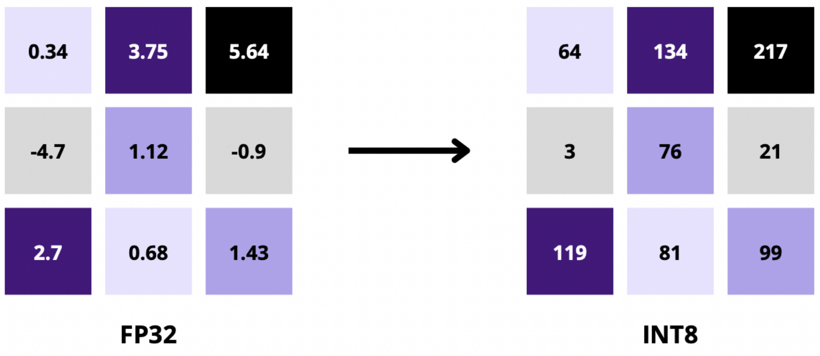

Figure 3: Example of a weight matrix in FP32 being mapped to INT8.

There are two main categorizations of quantization techniques. The first one is based on the distribution of quantization levels, how the original parameters are mapped to a lower precision representation. This includes uniform, non-uniform, and mixed-precision quantization. The second one is based on the timing and methodology of the quantization process. This includes static quantization, dynamic quantization, and quantization-aware training.

2.2.1 Uniform Quantization

Uniform quantization is a straightforward approach in which the range of parameter values is divided into equally spaced intervals, or "bins", and each interval is assigned a representative value. Parameters are then replaced by their respective representative values, resulting in a reduction in bit representation. Uniform quantization can be further divided into scalar and vector quantization. Scalar quantization [7] involves quantizing each parameter independently, which can be computationally efficient but may not always yield the best compression-performance trade-off. Vector quantization [14] quantizes groups of parameters jointly, typically leading to better performance preservation at the cost of increased complexity.

2.2.2 Non-Uniform Quantization

Non-uniform quantization assigns variable-length intervals to parameter values, with smaller intervals assigned to regions with higher parameter density. This approach can better preserve the model's performance, as it allows for a more accurate representation of the parameter distribution. However, non-uniform quantization is generally more computationally demanding than uniform quantization. One popular non-uniform quantization technique is k-means quantization [6], in which the parameter space is clustered using the k-means algorithm, and each parameter is replaced by the centroid of its corresponding cluster.

2.2.3 Mixed-Precision Quantization

Mixed-precision quantization involves using different bit representations for different parts of the model, based on their importance to the overall performance. This approach can strike a balance between model size reduction and performance preservation by allocating more bits to critical parameters and fewer bits to less important ones. Mixed-precision quantization [11, 39] can be applied at various levels of granularity, including per-layer, per-group, or even per-parameter. Adaptive techniques have also been proposed to determine the optimal bit allocation for each part of the model automatically.

2.2.4 Quantization Algorithms

The second categorization of quantization refers to the timing and methodology of the quantization process, which includes static quantization, dynamic quantization, and quantization-aware training. Static quantization is a post-training technique that applies quantization to a trained model without any further modifications. This approach is computationally efficient but may result in performance degradation due to quantization errors. Dynamic quantization, on the other hand, applies quantization during the inference process, adjusting quantization levels based on the input data's characteristics. This method allows for better performance preservation but can be computationally demanding. Quantization-aware training incorporates the effects of quantization during the training process, allowing the model to adapt to the reduced precision representation. This approach can help mitigate performance degradation caused by quantization and improve the model's performance on quantization-friendly hardware platforms.

2.3 Knowledge Distillation

Knowledge distillation is a model compression technique that involves training a smaller, more efficient model (student) to mimic the behavior of a larger, more accurate model (teacher). The central idea behind knowledge distillation is to transfer the knowledge embedded within the teacher model to the student model, enabling the latter to achieve comparable performance with fewer parameters and reduced computational complexity. The student model is trained using a combination of the original data and the outputs from the teacher model, with the aim of transferring the knowledge from the teacher to the student while reducing the model size. This subsection provides an overview of the main knowledge distillation approaches, along with their respective strengths, weaknesses, and recent advances.

2.3.1 Basic Knowledge Distillation

The basic knowledge distillation approach involves training the student model using a combination of the original data and the outputs (soft targets) from the teacher model. The soft targets, which are the probabilities produced by the teacher model before applying the final activation function, provide a richer source of information compared to the hard targets (ground truth labels) and can effectively guide the student model towards learning the teacher's behavior.

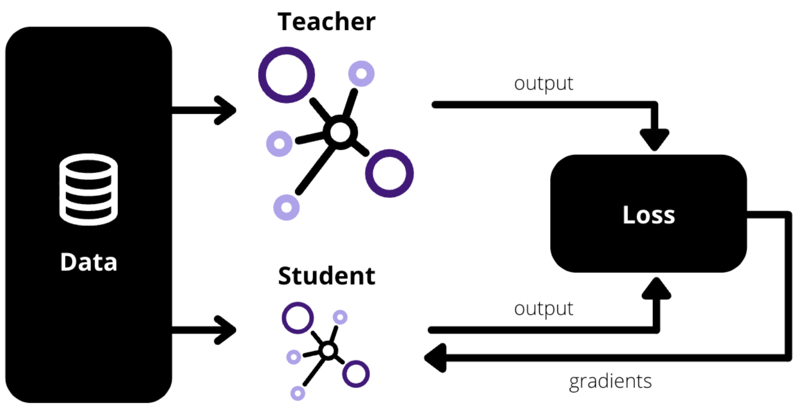

Figure 4: Example of a basic knowledge distillation process.

The student model is trained by minimizing a weighted combination of the traditional loss, usually Cross-Entropy loss, with respect to the hard targets, and the distillation loss with respect to the soft targets. The distillation loss is often calculated using the Kullback-Leibler (KL) divergence between the teacher's and student's probability distributions

2.3.2 Advanced Distillation Techniques

Over the years, several advanced knowledge distillation techniques have been proposed to improve the effectiveness of knowledge transfer between the teacher and student models. These techniques can be broadly classified into three categories: intermediate representation distillation, self-distillation, and ensemble distillation.

Intermediate representation distillation [23] involves transferring knowledge from intermediate layers of the teacher model to corresponding layers in the student model. This approach can help the student model learn more effectively by leveraging the hierarchical representations learned by the teacher model. This category includes techniques such as feature map distillation and attention map distillation.

Self-distillation [48] refers to the process of distilling knowledge within the same model by training it multiple times with different temperature settings, where temperature is a hyperparameter used to control the smoothness of the soft targets. Higher temperatures lead to smoother probability distributions, which can facilitate more effective knowledge transfer.

Ensemble distillation [16] involves training the student model to mimic the behavior of an ensemble of teacher models. This approach can improve the student model's performance by leveraging the diversity and complementary strengths of multiple teacher models.

2.4 Summary

This section provided an overview of compression techniques for deep learning models, focusing on three main categories: pruning, knowledge distillation, and quantization.

Pruning aims to remove redundant or less important parameters from deep learning models. It can be broadly classified into structured and unstructured pruning. Unstructured pruning involves the removal of individual parameters, while structured pruning refers to the removal of entire structures within the model.

Quantization is a technique that reduces the precision of the parameters in a deep learning model by representing them with fewer bits. They can be categorized on the different distributions of quantization levels and how quantization is applied.

Knowledge distillation is a technique that involves training a smaller, more efficient model (student) to mimic the behavior of a larger, more accurate model (teacher). Basic knowledge distillation uses a combination of the original data and the outputs from the teacher model, while advanced distillation techniques include intermediate representation distillation, self-distillation, and ensemble distillation.

3 Related Work

This thesis proposes a tool that automates compression of models using a tailored selection of different compression techniques. In this section, we look at related work that aims to achieve the same objectives and discuss how our tool differs. We divide this section into three parts: research that combines different compression techniques, research that automates compression, and how this thesis compares to the other research.

3.1 Combined Compression Techniques

Recent research has shown that combining pruning, distillation, and quantization techniques can significantly reduce the size and number of computations of neural networks without affecting their accuracy. In this section, we cover some techniques that combine two or more complementary compression techniques to achieve higher compression rates.

3.1.1 Pruning and Quantization

The combination of pruning and quantization capitalizes on their complementary strengths: pruning reduces redundancy in neural connections, while quantization minimizes the precision required for weight representation.

HFPQ [12], introduced by Fan et al., applies channel pruning and quantization using power of 2 exponentials. It retrains the network after the pruning and quantization phases. Zhang et al. [50] proposed a method that jointly applies uniform quantization and unstructured pruning methods to both the weights and activations of deep neural networks during training. Guerra et al. [18] examined a combination of quantization and pruning techniques to achieve further network compression. Han et al. [20] introduced "deep compression", a three-stage pipeline that prunes the network by learning only the important connections, quantizes the weights and activations, and applies Huffman coding to further compress the network.

3.1.2 Pruning and Distillation

Pruning is effective at reducing overparametrization and model complexity. Distillation, on the other hand, complements this by transferring learned patterns from larger models, ensuring that these pruned, leaner models actually retain high-level performance.

Wang et al. [46] introduced a compression scheme that combines pruning and knowledge distillation methods by applying unstructured pruning to the teacher model. The aim of this is to avoid overfitting by the teacher model and generalizing the abstract knowledge it contains. Aghli et al. [1] combine weight pruning and knowledge distillation to compress ResNet models without altering the structure of the network. They apply weight pruning on a selected number of layers and then apply knowledge distillation on the remaining unaltered layers.

3.1.3 Quantization and Distillation

Quantization and distillation complement each other effectively: while the former minimizes the parameters' footprint, the latter ensures the retention of critical predictive knowledge within these leaner models, thereby creating high-performing, resource-efficient models.

Polino et al. [40] apply knowledge distillation to quantized student models. They incorporate the distillation loss in the original training loop. They also introduce differentiable quantization which configures the quantization points based on the behaviour of the teacher model. Zhang et al. [49] apply ternary quantization to BERT to achieve high compression rates. Because of the ultra low-bit representation knowledge distillation is applied to avoid the accuracy degradation.

3.1.4 Pruning, Distillation, and Quantization

There is also a range of research that combines all three combination categories. The ways these techniques are combined can differ in many ways.

Kim et al. [26] introduce PQK, which combines pruning, quantization and knowledge distillation. They use iterative weight pruning and quantization-aware training to create a smaller model. They add the removed weights from the pruning phase, back to the smaller model to create a teacher model, which is then used for knowledge distillation. Zhao et al. [51] use filter pruning to create a small student model, which is then distilled using a custom distillation loss and the original model. After distillation the weights are quantized and the model is fine-tuned. Kim et al. [25] present QRPK, which uses all three techniques simultaneously in an iterative approach. While the model is pruned based on the magnitude of its weights, they adapt the quantization function on the distribution of the remaining weights. Concurrently they use a distillation loss to finetune the model.

3.2 Automated Compression

AutoML, or automated machine learning, is a set of techniques and tools that automate the process of designing and optimizing machine learning pipelines. The compression of deep learning models has also been researched in this context. Research in this field mainly focuses on automatically exploring the design space and finding the optimal compression strategy

For example, AutoML for Model Compression (AMC) [22] is a technique that uses reinforcement learning to explore the design space and find the optimal pruning strategy. AMC has been shown to achieve state-of-the-art compression rates while maintaining high accuracy. Wang et al. [45] also apply reinforcement learning, in the form of an actor-critic method to find the optimal set of weights to prune.

Other automated compression frameworks focus on an iterative approach, taking into account the effects of the compression actions and adapting their approach as the reductions increase. For example Pocketflow [47], which implements a set of different compression techniques and a hyperparameter optimizer to find the ideal configuration. It iteratively compresses and finetunes a candidate model, which is then evaluated to get a reward score for the optimizer. Gusak et al. [19] don't use any of the pruning, quantization or distillation categories, but automates compression low-rank approximations, also with an iterative approach.

Next to reinforcement learning and iterative based approaches, there is also research taking other approaches. One of which is AutoCompress [34] which generates a set of structured pruning samples, and has a heuristics-based automated agent evaluate the samples and decide the optimal pruning actions. Another one is Bayesian Automatic Model Compression [44] that uses non-parametric Bayesian methods to learn the optimal quantization bit-width.

While the use of reinforcement learning to improve the compression of deep learning models is promising and iterative approaches can yield robust solutions, this thesis focuses on a more conventional approach. The tools uses a pipeline-like architecture relying on rule-based policies and combinations of complementary techniques, with a partially iterative approach for the pruning phase.

3.3 Comparison

There are some similarities and differences between EasyCompress and the related work. Like some of the works described in subsection 3.1 this tool also makes use of different compression techniques, and it also uses an automated pipeline to achieve compression like some of the works described in subsection 3.2. However, there are three main differences in the focus of this tool.

The first main difference is that the tool takes into account the goal and constraints of the compression. Depending on the goal it selects different techniques, and depending on the constraints it adapts its configurations. The decision to focus on this aspect allows increases the applicability in real-world scenarios. The focus shifts from trying to achieve the maximal compression, to finding the least-destructive way to reach a goal.

The second main difference is that the tool focuses on generalizability between model architectures, without requiring deep knowledge about the model itself. The other techniques have a limited scope, and dive deep into the architectures of the models they are evaluated on. For example, HFPQ [12] experimentally examines the sensitivity of each individual filter before devising a final strategy. Aghli et al. [1] use pruning to construct a teacher model without redundant parameters, but still requires them to manually construct a student model by manually changing the layer sizes. By focusing on generalizability, the tool is useful for a larger range of models.

A third way the tool differs is its focus on real-world applicability, refraining from using techniques such as unstructured pruning that may require specific execution environments to realize the metrics reductions. While this limits the compression potential, it again facilitates the use of the tool. There exist automated methods that also use unstructured pruning, such as AutoCompress [34], but they focus solely on pruning.

4 Tool Architecture

This section describes the architecture of EasyCompress. It explains how the tool works and how it's underlying modules cooperate to select a set of compression actions and how they are automatically implemented.

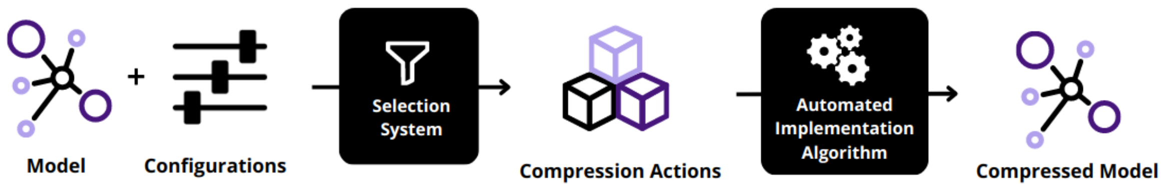

Figure 5: High-level overview of the tool's architecture.

EasyCompress consists of two primary, independent modules: an intelligent selection system and an automated implementation algorithm that consists of multiple compression modules. Figure 5 shows a high-level overview of how the tool works. The selection system computes various heuristics about the model. Based on these heuristics and the requirements provided by the user it devises a set of customized compression actions. These actions are then fed into the automated compression algorithm, consisting of the different compression modules, which applies them to the model.

The next subsections dive deeper into how the tool is designed and how it works.

4.1 Tool Workflow

Before the design of the architecture is discussed we clarify how EasyCompress is intended to be used. The goal of this thesis was to develop a tool that automates deep learning model compression with as little intervention as possible. The envisioned workflow of the tool goes as follows:

1. Provide a deep learning model, a corresponding dataset, and the necessary configuration parameters.
2. Specify the objective of the compression: is the goal to reduce model size, inference time, or energy usage?
3. Set a performance threshold and a compression target. The performance threshold is a lower bound for performance under which the compressed model should not go. The compression target is the level of compression the tool will aim to achieve.

Once all these items are provided the tool analyzes the model, comes up with a compression strategy and performs it. The next sections explain how the tool accomplishes this.

4.2 Selection System

The selection system's goal is to produce a set of tailored compression techniques to achieve the defined compression goal. Figure 6 visualizes how the selection system operates.

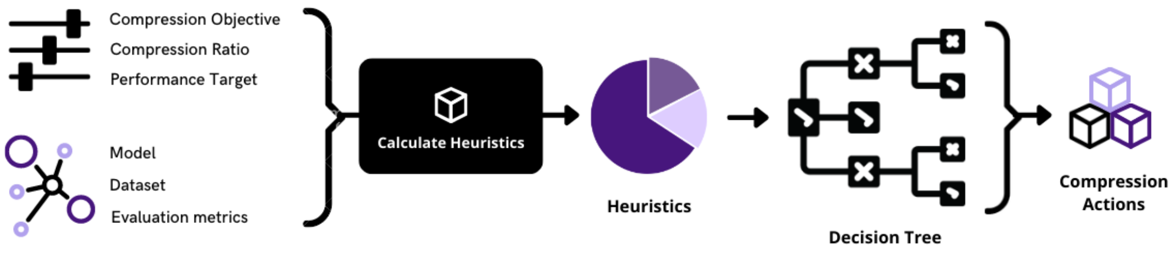

Figure 6: Overview of the selection system.

The selection system uses the configurations set by the user, such as the compression objective (model size, inference time, or number of computations), the compression ratio, and the performance target threshold. Based on the provided configurations and model it calculates a set of heuristics: the architecture type, the layer types, the layer distributions, the parameter distribution, etc. Based on these heuristics a decision tree algorithm selects the appropriate compression actions and then tailors their configuration to meet the compression target. The decision tree consists of three modules, each modeling the selection procedure for one of the compression categories: pruning, distillation, and quantization. The selection procedures for these three categories are explained in the following subsections.

4.2.1 Heuristics

This section provides an overview of the heuristics generated and/or used by the selection system and details their respective purposes. Heuristic are generated from the model, dataset, and configurations details.

- **Architecture type**: the architecture of the model, e.g. FNN, CNN, or transformers. Used for pruning technique.
- **Layer types**: a list of all the layer types used in the model. Used for pruning strategy.
- **Layer distribution**: a list of the counts of each layer type. Used for pruning strategy.

- **Parameter distribution**: a list of counts of the number of parameters belonging to each layer type. Used for pruning strategy.
- **Performance metric**: the metric that is used to evaluate the performance of the model, e.g. accuracy, f1-score, or perplexity. Used for distillation.
- **Compression Objective**: the objective of the compression: model size, inference time, or energy consumption. Used for pruning technique.
- **Compression Target**: the desired amount of compression. Used for configuring pruning technique.
- **Performance Threshold**: the required performance threshold the model should still achieve. Used for all three techniques.
- **Computational Resources**: the availability of computational resources to perform compute-heavy operations on the model. Used for distillation and quantization.
- **Target Backend**: the backend to which the compressed model will be deployed, either GPU or CPU. Used for quantization.

The next three subsections explain further how these heuristics are applied in the selection procedures.

4.2.2 Pruning Selection

The pruning technique focuses solely on structured pruning, but still requires a lot of factors to be tailored correctly. There are two primary decisions to make: determining a pruning strategy and selecting a pruning technique.

The pruning strategy decides what layers should be pruned. There are four pruning strategies: "Linear", "Convolutional", "Attention", and "Global". The first three strategies focus pruning efforts on the respective layers. The last strategy "Global" takes into account all the layers for pruning.

When the compression objective is model size, the tool fist analyses the parameter distribution of the types of layers. Next to that it also calculates an estimate of the number of parameters that should be pruned based on the given compression target. It then follows a greedy strategy where it focuses pruning on the layer types that represent the largest amount of parameters. Based on this the focus of the pruning action tends to lie on pruning linear layers, since these tend to have the majority of parameters.

When the compression objective is inference time or energy usage the focus lies on reducing the number of computations. The strategy then focuses on pruning of layers that are compute-intensive, such as convolutional layers for CNNs or self-attention layers for transformers, since these account for the majority of the computations. It calculates an estimate for the flops per layer and checks whether the flops reduction can be achieved with a selection of layers. If this can't be reached without pruning more than 80% of parameters the selection, it uses all layers. Empirically I found that performance recovery potential drops significantly when passing this threshold.

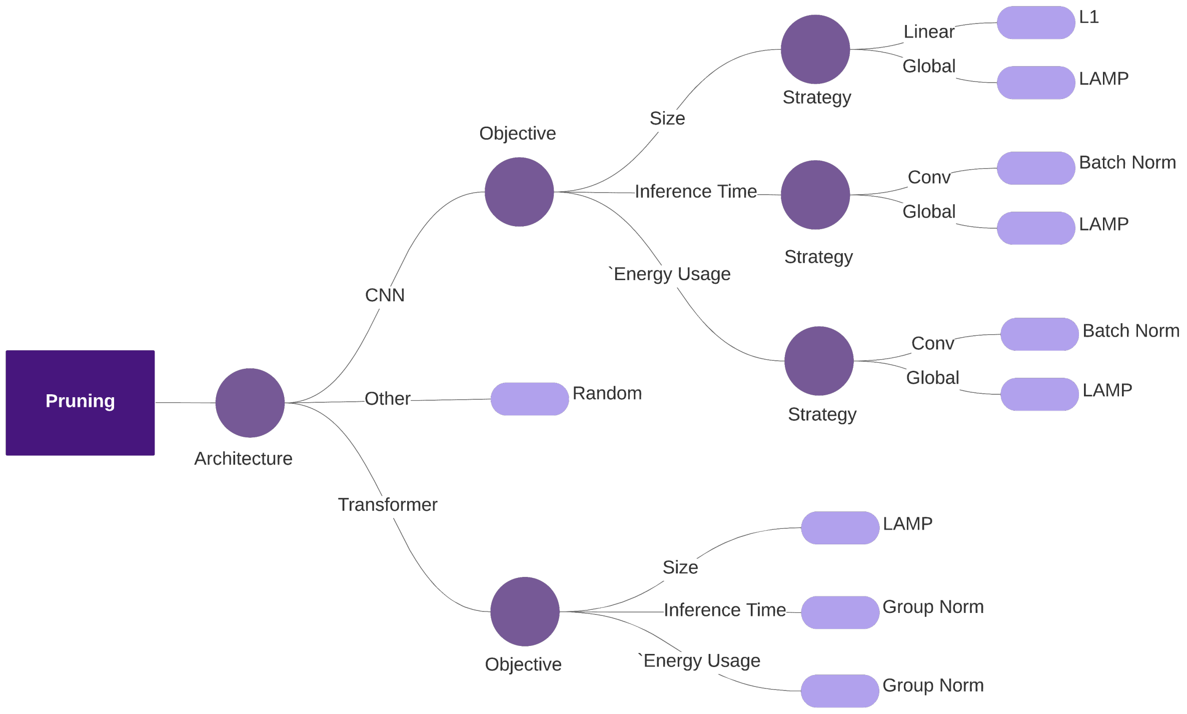

Figure 7: Visualization of the pruning decision tree.

As described in subsection 4.4 the pruning module offers 5 different pruning techniques: Random, Magnitude, Batch Norm Scale, Group Norm, and LAMP pruning. The selection of a proper pruning technique focuses on 3 factors: the architecture, the compression objective, and the selected compression strategy. It goes over a decision tree, visualized in Figure 7, to find the right pruning technique based on these factors. Overall the LAMP technique is effective among all objectives, the L1 technique tends to be preferred when the strategy targerts linear layers, SLIM is useful when the strategy is focused on convolutional layers, and GroupNorm is preferred for transformers.

4.2.3 Distillation Selection

The distillation phase of the tool has two main goals: recovery of performance, and avoiding overfitting that can be caused by normal finetuning. The selection of the proper distillation technique is based on two factors: availability of computational power and the nature of the evaluation task.

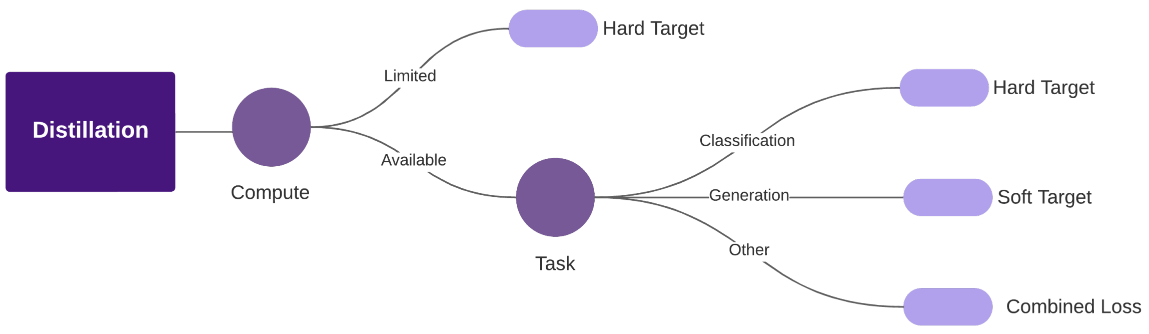

Figure 8: Visualization of the distillation decision tree.

Hard-target distillation is chosen for tasks prioritizing high confidence and precision, such as accuracy, and is suitable for robust student architectures. It is also the go-to method when the computational resources are limited and the distillation should converge fast. Soft-target distillation is ideal for simpler models and tasks needing nuanced class probability understanding. Combined distillation balances precision and class understanding, fitting a variety of architectures and performance targets, and is often optimal when high recovery from teacher model's outputs is needed.

4.2.4 Quantization Selection

For the quantization technique a few factors influence the decision: the target backend, the availability of computational resources, and the permitted performance drop. The decision tree is visualized in Figure 9.

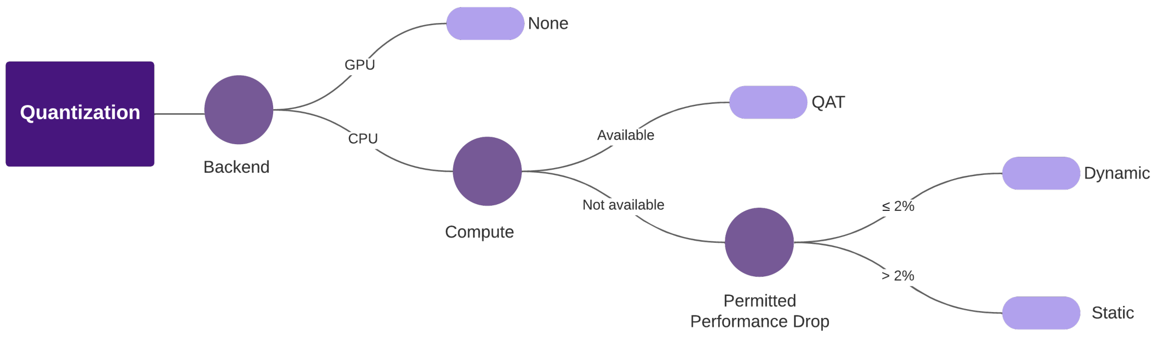

Figure 9: Visualization of the quantization decision tree.

When the compressed model is to deployed on a GPU, the quantization is omitted. The tool currently only supports quantized models to be runnable on CPUs. When there is no resource-constraint, QAT is selected, since it generally better preserves performance. If retraining is not possible, because of a resource-constraint, we focus on post-training quantization techniques. The decision between dynamic and static quantization is made based on how much the performance is permitted to drop. When a larger reduction in performance is tolerated, static quantization is selected, since it quantizes both weights and activations, and leads to a higher compression rate. On the other hand, when the performance isn't allowed to drop significantly, dynamic quantization is selected. This better preserves accuracy but leads to lower compression rates because not all parameters are quantized. The threshold value to decide between the two is set at 2%.

4.3 Automated Implementation Algorithm

The second main module of the tool is the automated implementation algorithm. It is responsible for accepting a set of tailored compression actions and implementing them. The general compression workflow is depicted by Figure 10. It exists of three phases, one for each compression category.

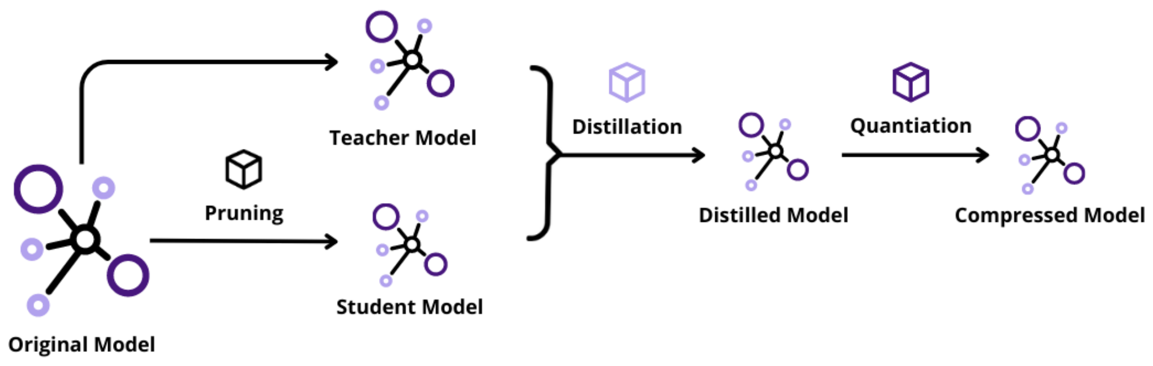

Figure 10: Overview of the automated implementation algorithm.

First, the model is pruned to create a custom student model, a scaled-down version of the original model targeted for compression. See subsection 4.4 for how it works, what pruning techniques and strategies are offered, and how they are implemented. Using pruning to create a student model allows for automatically designing a smaller model architecture that maintains the strongest features of the teacher model.

Next, we use the knowledge distillation compression action with the original model as a teacher model. See subsection 4.5 for how it works, what distillation techniques are offered, and how they are implemented. Conventional knowledge distillation approaches require users to design a separate student model, but this tool automates the process by utilizing pruning techniques to generate a custom student model for distillation. If computational resources are unavailable, the tool can bypass the distillation and fine-tuning phases, proceeding directly to quantization.

Lastly, the quantization action is applied to further compress the student model. See subsection 4.6 for how it works, what quantization techniques are offered, and how they are implemented. For the post-training quantization techniques, such as static and dynamic quantization, this happens after the distillation phase. For quantization-aware training, the quantization phase is applied in parallel with the distillation phase.

4.4 Pruning Module

The goal of the pruning module is to remove the least important structures in the model. After pruning we should have a model that contains the core architectural features, so that it can be used by the distillation module as a student model. As described in subsection 2.1

unstructured pruning doesn't maintain regularity. A key objective of this thesis was to develop a tool that could compress models while still ensuring they remain runnable on standard hardware. For this reason the pruning module focuses on structured pruning techniques, as they don't require specialized hardware or execution environments to capitalize on the reductions.

4.4.1 General Mechanism

Pruning structures changes the architecture directly. This has as benefit that the pruning has direct effects which can be measured in model size, inference time, and number of computations. The challenge with the changing architecture is that it easy to break the model. Many layers are dependent on each other and have specific dimension requirements for the input they receive, we call this interdependent structure groups. For more complex architectures the effects of a change to one layer can propagate through many other layers, which in their turn need to be adapted. These groups are highly dependent on the architecture of the model, for this reason most structured pruning techniques use manually-designed, architecture-specific grouping schemes, which are non-generalizable to new architectures.

For this reason this tool uses a structured pruning strategy that uses a dependency graph, proposed by Fang et al. [13], which removes structurally-grouped parameters from neural networks in a architecture-generalizable way. It does this by decomposing the network into paramaterized layers, and non-parameterized operation, modelling their dependencies, and grouping these components based on these dependencies. These groups can then be used as pruning candidates, on a variety of criteria.

The key benefit of this approach is that we can use the same set of pruning techniques for different model architectures, which corresponds with our objective of building a system that can be used on models from a diverse set of architectures. The structures targeted by the pruning module are channels, also known as dimensions of intermediate features. The interpretation of a channel can vary per layer type; for linear layers as neurons, for convolutional layers as filter channels, for an output layer as predictions.

4.4.2 Techniques

The pruning module offers five basic pruning strategies for selecting the groups to be pruned, based on the importance criterion: Random Pruning, Magnitude Pruning [29], BN Scale

Pruning [37], Group Norm Pruning [13] and LAMP pruning [28].

Random Pruning randomly removes network weights or channels without considering their magnitude or importance. It generally leads to surprisingly robust performance.

Magnitude pruning is a straightforward yet effective method that prunes the portion of the parameters of the model with the lowest L1-norm absolute value. The underlying assumption is that smaller weights have a lesser impact on the output of the model, and hence can be removed without significant loss of performance.

Batch Norm Scale Pruning uses the values of batch normalization layers in a network as a scaling factor. Batch normalization layers is used in neural networks to stabilize and accelerate the training process by normalizing the activations of a given layer. It helps reduce internal covariate shift, allowing for faster convergence, better generalization, and the use of higher learning rates in neural networks. [37]

Group Norm Pruning constructs groups of interdependent parameters and assigns a group-norm to each of them, which is the average magnitude of the whole group. It then prunes the groups with the lowest group-norms. [13]

LAMP Pruning uses a LAMP score as pruning criterion. This is a rescaled version of the weight magnitude that approximates the model-level L2 distortion, which is the Euclidean distance between the output of the layer before pruning and the output of the layer after pruning. [28]

4.4.3 Implementation

Code for the pruning module is available on GitHub[1]. The module's implementation is centered around a main method, `structure_pruning`, supplemented by a collection of helper methods and functions for handling the different aspects of the pruning process.

The `structure_pruning` method is the entry point and implements the core process of pruning the neural network model. It starts by preparing the model for the pruning process, which includes preparing the model for execution on the chosen device, generating example inputs for the model, and setting up the optimizer.

[1]https://github.com/abel-vs/thesis/blob/main/src/compression/pruning.py

For deciding which layers to prune, the method offers the possibility to pass a custom set of prunable layers. If none are provided, the function will automatically generate a set of layers that should not be pruned using the `get_layers_not_to_prune function`. This function determines the layers not to prune based on their position in the model (e.g. first and last layer) and their interdependencies with other layers. These include sensitive layers involved in skip connections or shortcut paths, batch normalization layers, and layers with unique roles in the model's architecture such as the input and output layers.

After the layers that should be ignored during pruning are identified, the pruner object is created. The `get_pruner` function provides a way to instantiate different pruners based on the pruning technique chosen. It supports several pruning methods, including Random Pruning, Magnitude Pruning, LAMP Pruning, Group Norm Pruning, and Batch Normalization (BN) Scale Pruning.

The `structure_pruning` function then iteratively applies the pruner to the model. After each pruning step, the model can optionally be finetuned, which serves to recover any performance lost due to the removal of parameters.

The `calculate_channel_sparsity` function is another critical part of the module. It estimates the required channel sparsity level to reach a specified target global sparsity. This function supports Conv2D and Linear layers, and it calculates the target sparsity level based on the proportion of convolutional parameters in the total parameters of the model.

4.5 Knowledge Distillation Module

The goal of the distillation module is to recover the performance of the model, after it has been compressed by the pruning module. The distillation module focuses solely on logits-based distillation procedures. This decision is rooted in the principle of flexibility that is an objective of the tool. In the context of model distillation, flexibility signifies the freedom to construct student models based on what the pruning module has creates, regardless of the architecture or complexity of the teacher model. Focusing on logits-based approaches leads to more freedom in what can, and what can not be pruned.

The distillation module offers three types of logits distillation techniques: soft-target, hard-target, and combined distillation. Each of these techniques allows for a different level of flexibility in modeling the output distribution of the teacher model.

4.5.1 Techniques

Soft-target distillation is a technique that leverages the probability distribution over class labels generated by a pre-trained teacher model to train a smaller student model. Instead of using the hard ground-truth labels, this approach takes into account the teacher model's confidence in each class. This allows the student model to learn a richer representation of the input data, as it also captures the teacher model's knowledge about the relationships between different classes. It is calculated as follows:

$$S = \frac{1}{T^2} \cdot \text{KL\_Divergence}(\text{Softmax}(z_{\text{teacher}}/T), \text{Softmax}(z_{\text{student}}/T)) \tag{1}$$

Hard-target distillation uses the ground-truth labels as targets for training the student model. The student model is trained to minimize the difference between its own class predictions and the hard labels, typically using a loss function like cross-entropy. This approach is more straightforward compared to soft-target distillation but may not capture the teacher model's knowledge about the relationships between classes as effectively. It is calculated as follows:

$$H = \text{CrossEntropy}(y, \text{Softmax}(z_{\text{student}})) \tag{2}$$

Combined distillation is an approach that blends the advantages of both soft-target and hard-target distillation techniques. In this method, the student model is trained using a combination of the teacher model's probability distribution (soft-targets) and the ground-truth labels (hard-targets). The loss function is a weighted sum of the soft-target loss (e.g., KL divergence) and the hard-target loss (e.g., cross-entropy). It is calculated as follows:

$$C = \alpha H + (1 - \alpha) S \tag{3}$$

By balancing the contributions of both soft and hard targets, combined distillation allows the student model to benefit from the more nuanced representation provided by the teacher model while still being guided by the ground-truth labels.

These techniques have distinct advantages and trade-offs. Soft-target distillation offers improved generalization, robustness to label noise, and better calibration of predicted probabilities, making it useful for tasks with complex or noisy input data. Hard-target distillation is simpler and more straightforward but may not capture the teacher model's knowledge as effectively. It is preferred for tasks like classification. Combined distillation aims to strike a balance between these two methods, offering a versatile approach for a wide range of tasks.

4.5.2 Implementation

Code for the distillation module is available on GitHub[2]. The implementation of the distillation techniques exist of a main `distillation_train_loop` method that consists of two modes.

The first mode is based on a given threshold that acts as a target score, where the method distills the model until the evaluation metric reaches this target on the validation set. The second mode is based on a given number of epochs that the method distills the model. This mode is targeted towards scenarios where computational resources are limited.

This method is provided with the teacher model, student model, one of the three specific distillation techniques, and some customizable parameters such as a distillation criterion to calculate the loss during distillation training, an optimizer, and a threshold or a number of epochs.

Both modes contain an adaptive early stopping criterion, which for each epoch checks whether the validation score is still increasing. The method tracks the best performing model and its corresponding score. The early stopping uses a customizable patience with a default value of 3, which stands for the allowed number of epochs the best validation score hasn't improved.

The actual distillation strategy is indicated by the `technique` parameter, which links to one of the three distillation techniques: `soft_target_distillation`, `hard_target_distillation`, and `combined_loss_distillation`. These techniques are defined as separate functions, each implementing their distillation procedures. The distillation criterion can be configured by the selection system, depending on the specific task and evaluation method.

[2] https://github.com/abel-vs/thesis/blob/main/src/compression/distillation.py

4.6 Quantization Module

As described in subsection 2.2 the goal of quantization is to represent weights in a lower number of bits thereby leading to a smaller storage size and simpler computations. The tool offers three different approaches: dynamic quantization, static quantization, and a quantization aware training. The static and dynamic quantization techniques are post-training quantization techniques, meaning they don't have a fine-tuning stage. These techniques can be selected when the accuracy-constraint allows for a decrease, or when fine-tuning the model again is too resource-intensive. Quantization-aware training is overall the best to maintain accuracy.

4.6.1 Techniques

Static quantization is performed after the model has been trained. It quantizes both the weights and activations of the model using a calibration step on the training partition of the given dataset. This technique is best suited for models with simpler structures, such as CNNs, and cases where retraining is not feasible or desired, for example because of resource-constraints. Static quantization offers a significant reduction in model size and inference time. However, the accuracy loss can be higher compared to dynamic quantization and QAT.

Dynamic quantization is also a form of post-training quantization. It only quantizes the weights of the model, while activations remain in floating-point format. Quantization is performed during inference and is adjusted dynamically based on the input data. It is best suited for models with complex control flow and recurrent structures such as Transformers, where quantizing activations may be challenging. Dynamic quantization also doesn't require retraining, making it ideal for cases with resource-constraints. Dynamic quantization results in smaller accuracy losses compared to static quantization. However, the reduction in model size and acceleration in inference time tends to be smaller compared to static quantization and QAT.

Quantization-Aware Training (QAT) simulates quantization during the training process, allowing the model to adapt to the quantization effects. This technique quantizes both the weights and activations of the model and is best suited for models where maintaining the highest possible accuracy is crucial and retraining is acceptable to optimize the model for quantized deployment. QAT results in a significant reduction in model size and inference time and typically has better accuracy compared to static quantization, as the model is trained to be robust to quantization effects. However, it requires retraining the model, making it more time-consuming to implement than dynamic and static quantization.

4.6.2 Implementation

Code for the quantization module is available on GitHub[3]. The three quantization techniques are implemented by three separate methods, which utilize the quantization methods from the PyTorch quantization library.

Static quantization is implemented using the torch.quantization.quantize function, which takes care of calibration, quantization, and dequantization. Calibration is performed using the `calculate_qparams` method, which determines the optimal scaling factor and zero-point values. The quantization and dequantization processes are handled by the QuantStub and DeQuantStub classes.

Dynamic quantization leverages the `torch.quantization.quantize_dynamic` function. The module first quantizes the model's weights using the `torch.quantization.default_dynamic_qconfi` configuration. During the inference, activations are quantized dynamically with the help of the of custom layers such as `nn.quantized.dynamic.Linear` and `nn.quantized.dynamic.LSTM` classes for linear and recurrent layers, respectively.

Quantization-Aware Training (QAT) incorporates fake quantization modules into the model using the `torch.quantization.prepare_qat` and `torch.quantization.convert` functions for preparation and conversion, respectively. Under the hood, the `FakeQuantize` class is used for simulating quantization effects during training, and at conversion the modules are converted to quantized layers such as the `nn.quantized.QConv2d` or `nn.quantized.QLinear` layers.

[3]https://github.com/abel-vs/thesis/blob/main/src/compression/quantization.py

5 Experimental Setup

In the opening section of this thesis, I introduced the primary research questions and objectives that guided the research and development of EasyCompress. This section presents the corresponding experiments designed to assess the successful achievement of these objectives. This section gives an overview of how the experiments are set up, what models, datasets, and metrics are used for evaluation, and how the experiments were implemented.

5.1 Overview of the experiments

The goal of the experiments is to evaluate the performance of the tool and to answer the research questions defined in the introduction. For the 5 research questions we applied the following approaches to validate their achievement:

1. **How can the compression process be automated?**
   To answer this, we go over the architectural design choices to automate the process. We then compare the manual steps required with and without the automated process to quantify the time and effort saved. The results can be found in subsection 6.2 and are further discussed in subsection 7.2.

2. **How can compression techniques be combined effectively?**
   To answer this, we examine the effects of applying the different compression methods. We analyze the added influence of each module and validate whether combining compression methods leads to better reductions. The experiments and results can be found in subsection 6.1 and are further discussed in subsection 7.1.

3. **How can compression techniques be tailored to meet specific compression and performance requirements?**
   This question was answered in section 4 where we explained how the tools selects and configures compression actions to meet the requirements.

4. **How does the compression goal impact the selection of techniques?**
   To answer this, we analyze the inner workings of the selection system we designed. The main focus is put on the impact of the compression goal. For this question, no experiments were needed, so the analysis can directly be found in subsection 7.3.

5. **How well does the tool's selected compression set preserve accuracy for different compression goals?**
   To answer this, we evaluate the tool on a diverse set of models and compression configurations and analyze how well the accuracy is preserved. For each compression objectives we set three targets and observe the reductions. The experiments and results can be found in subsection 6.3 and are further discussed in subsection 7.4.

5.2 Models and Architectures

The tool is designed with the two major types of architectures in mind: Convolutional Neural Networks and Transformers. These two architectures have been the dominant state-of-the-art models for Computer Vision and Natural Language Processing respectively.
For the CNNs, I selected ResNet50, VGG-16, and EfficientNet. These three models are popular CNNs developed by different research groups, each with unique features that make them ideal for testing:

- **ResNet50** is a 50-layer deep residual network developed by Microsoft Research. Its 50-layer deep architecture and residual connections help alleviate the vanishing gradient problem. It offers excellent performance in image classification tasks and lower computational complexity.
- **VGG-16** is a 16-layer deep convolutional neural network developed by the Visual Geometry Group at the University of Oxford. VGG-16 has 16 layers and small (3x3) convolutional filters. It performs strongly in image classification and object recognition, but has a higher computational cost due to its depth and parameter count.
- **EfficientNet-B4** is the median network of the EfficientNet family, developed by by Google Research. It uses a compound scaling method for better performance and resource utilization. EfficientNet-B4 achieved state-of-the-art results in computer vision tasks while maintaining computational efficiency.

To test the generalizability of the tool we also tested a Transformer architecture:

- **BERT** is a bidirectional Transformer developed by Google Research, designed for pre-training on large-scale text corpora. Its bidirectional context encoding allows it to excel in various natural language understanding tasks, such as question answering and sentiment analysis.

5.3 Datasets

The CNN models are evaluated on three popular datasets: MNIST [9], CIFAR-10 [27], and ImageNet-1k [8]. The transformers are evaluated on the SQUAD [41] dataset.

- **MNIST** is a database of 70,000 28x28 grayscale images of handwritten digits. Due to its simplicity and relatively small size, it is often used as a baseline for image recognition and machine learning algorithms.
- **CIFAR-10** is a collection of 60,000 32x32 color images in 10 classes, with 6,000 images per class. It is commonly used as a benchmark dataset for image classification tasks.
- **ImageNet-1k** is a subset of the larger ImageNet[8] dataset that includes 1,000 categories and over 1.2 million images. It is also commonly used for benchmarking image classification models.
- **SQuAD** (Stanford Question Answering Dataset) is a large benchmark dataset for question-answering tasks. It consists of real-world articles paired with 100,000+ questions and answers. It is widely used for the evaluation and benchmarking of machine reading and comprehension algorithms.

5.4 Evaluation Metrics

This section describes the metrics used to assess the performance and degree of compression of the compressed models.

The evaluation metrics for the compression degree of the model are described in three ways. Model size is measured in megabytes (MB), inference time in milliseconds (ms), and number of computations in floating-point operations (FLOPS). Reductions are reported as X-fold reductions, which relates to a percentual reduction as follows: an X-fold reduction means the final value is $\frac{1}{X}$ of the original, translating to a $(1 - \frac{1}{X}) * 100\%$ reduction. For example, a 2-fold reduction equals a 50% reduction.

Performance on the MNIST, CIFAR-10 and ImageNet-1k datasets is measured in accuracy, defined as the ratio of correct predictions made by a model to the total number of predictions. Performance on the SQUAD benchmark is measured using an F1-score. The F1-score is more suitable for tasks that have imbalanced classes or partial matches, such as SQUAD where a generated answer can match a reference answer only partially. F1-score is calculated as the harmonic mean between precision and recall:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4}$$

The F1 score in the context of the SQuAD dataset is computed using a slight variation of the usual definition of precision and recall. Given a predicted answer and a ground truth answer, both represented as sequences of tokens, it works as follows. Precision is the proportion of predicted tokens that exist in the ground truth answer. It measures how many of the tokens that the model predicted are actually relevant. Recall is the proportion of ground truth tokens that exist in the predicted answer. It measures how many of the relevant tokens were actually captured by the model's prediction.

5.5 Implementation Details

The tool is fully written in PyTorch. An interface to interact with the tool was written in Next.js and uses FastAPI to communicate with the backend.

The pretrained CNNs were imported from the 'torchvision' package and adapted based on the datasets and tasks. The pretrained Transformers were imported from the 'transformers' package from Huggingface and adapted based on the datatsets and tasks.

Experiments were conducted in a custom Docker container hosted on the Ronaldo compute server from the TU Delft. I made this container available on Docker Hub[4] for reproducibility. The experiments were processed on two NVIDIA GeForce RTX 3080 GPUs.

All code is available on Github[5] together with some example notebooks. I also developed an interface with which users can experiment with the tool in an approachable manner, it is available online at https://thesis.abelvansteenweghen.com and the code is also available on Github [6].

[4]https://hub.docker.com/r/abelvs/thesis
[5]https://github.com/abel-vs/thesis
[6]https://github.com/abel-vs/thesis-app

6 RESULTS

This section presents the results of the series of experiments performed to evaluate the performance of the tool and to answer the research questions defined in section 1. An overview of the experiments is given in subsection 5.1.

The results are presented in the following order: The effects of the individual compression modules and their combinations are analyzed in subsection 6.1. The automation algorithm is evaluated in subsection 6.2. Lastly, bust most importantly, the overall performance of the tool is evaluated on a range of models and compression configurations in subsection 6.3.

6.1 Compression Module Analysis

In this section we present the analysis of the compression modules. These experiments aim to answer how the compression techniques can be combined effectively. For every module we performed some exploratory experiments to evaluate the impact and behaviour of each technique. How the techniques are combined is explained in subsection 4.3, this section focuses on validating that combining the compression techniques lead to better compression ratios. This section presents the setup and results of these experiments, the implications of these results are presented in subsection 7.1.

6.1.1 Pruning Module

To evaluate the effects of the pruning module, we compare the different pruning techniques on an example use case: pruning a ResNet-50 model with as compression objective a model size reduction of 50%, and evaluated on the CIFAR-10 dataset. We analyze two distinct behaviours: the behaviour of accuracy degradation that happens when pruning parameters, and the behaviour of the accuracy recovery when finetuning the pruned models.

The results are presented in the following two figures. Figure 11 compares the accuracy degradation of the different techniques. It plots the accuracy in function of the percentage of pruned parameters. Figure 12 compares the accuracy recovery of the different techniques on a pruned Resnet-50 model. It plots the accuracy in function of the number of finetuning epochs.

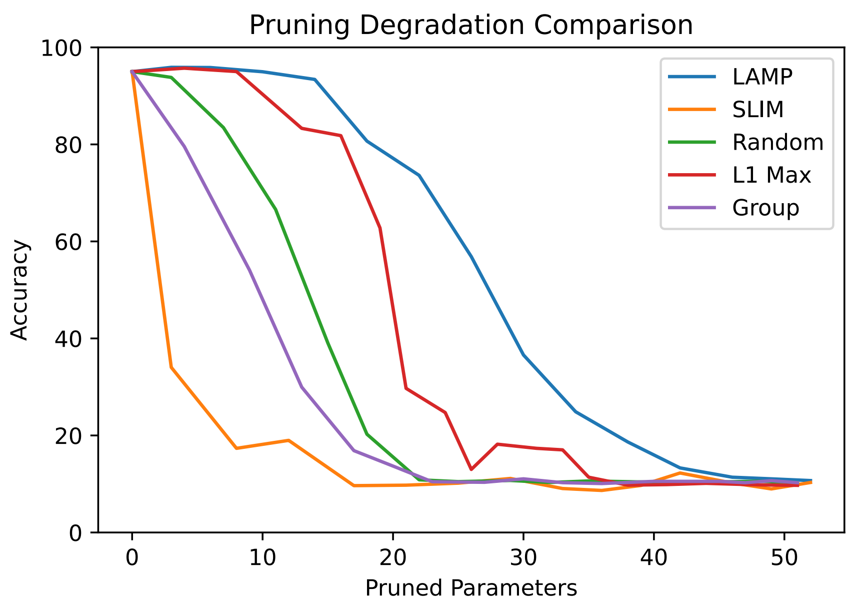

Figure 11: Comparison of accuracy degradation of pruning techniques on Resnet-50 and CIFAR-10.

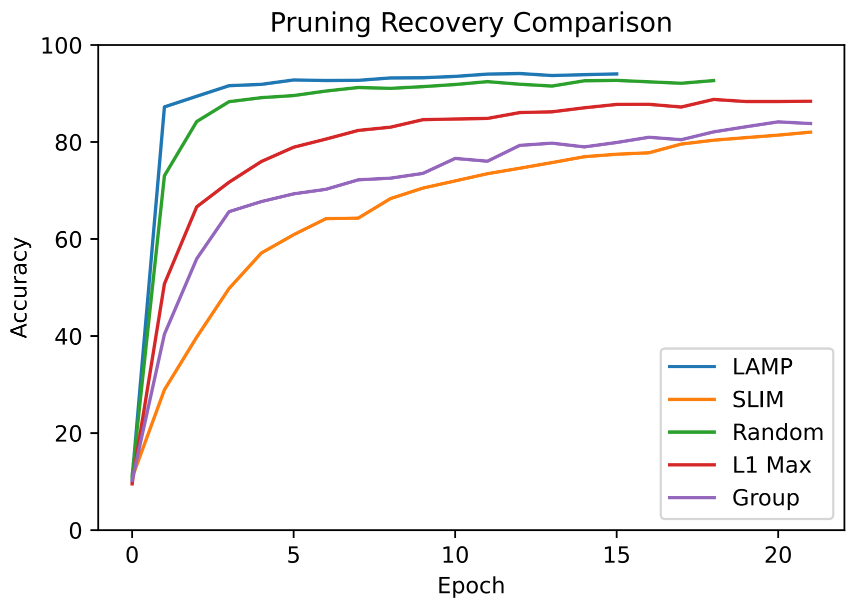

Figure 12: Comparison of accuracy recovery of pruning techniques on Resnet-50 and CIFAR-10.

6.1.2 Distillation Module

The benefit of combining distillation and pruning techniques comes from the ability to better recover accuracy compared to basic finetuning. To evaluate the effect of the distillation module we apply the different distillation techniques to a pruned model. As an example use case we again use a Resnet-50 model that has been pruned by 50% via the random pruning, and evaluated on the CIFAR-10 dataset. As a comparison baseline we also add the recovery graph from finetuning the model using the loss function used during training. Figure 13 plots the accuracy in function of the number of distillation/finetuning epochs.

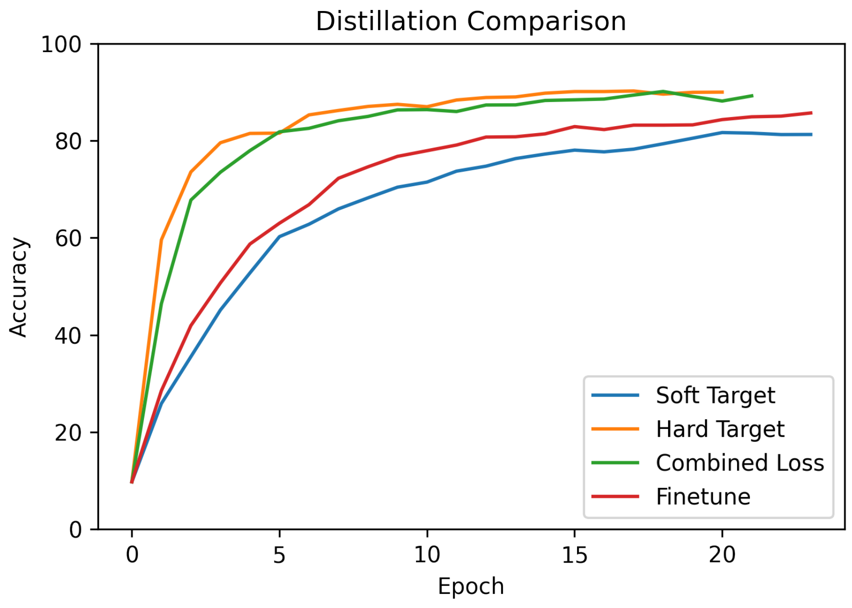

Figure 13: Comparison of different distillation techniques on Resnet-50 and CIFAR-10.

6.1.3 Quantization Module

To evaluate the effect of adding quantization to the the compression action set we evaluate the accuracy degradation effects of adding quantization on both an uncompressed and compressed version of ResNet-50. Table 1 presents the accuracy degradations of the different quantization techniques applied on the different models.

Quantization Type	Original	Compressed
Static Quantization	-1.24%	-1.33%
Dynamic Quantization	-0.68%	-0.62%
QAT	-0.28%	-0.29%

Table 1: Accuracy reductions of quantizing Resnet-50 for the CIFAR-10 benchmark.

6.1.4 Combined Reductions

To validate that combining different techniques can lead to more efficient compression we evaluate three models with three different compression action sets. The first compression relies solely on pruning, the second one combines pruning with distillation, and the third one uses all three techniques to achieve the compression goal. The results presented in Table 2 used as compression goal a model size reduction by 50%.

Model	Pruning	Pruning + Distillation	Pruning + Distillation + Quantization
Resnet-50	-2.4%	-1.5%	-0.8%
VGG-16	-1.2%	-0.8%	-0.6%
EfficientNet-B4	-2.5	-1.8%	-0.7%

Table 2: Accuracy reductions of combining compression techniques for the CIFAR-10 benchmark.

6.2 Automation Analysis

This section presents an analysis of the automation of compression pipeline. It aims to assess whether objective of compression automation has been successfully achieved. The design and implementation of the algorithm were explained in subsection 4.3. To quantify the degree of automation, this section presents a comparison in manual steps performed.

Table 3 lists a series of recurring steps that are part in the process of compressing a deep learning model. In a normal scenario, a researcher or developer has to go through all these steps to compress a model. The goal of developing the automated compression tool was to eliminate the need for manual intervention on most of these steps.

The tool automates 10 from the 16 listed manual steps. It has to be noted that the remaining manual procedures involve user decisions, including the selection of the model and dataset, as well as establishing the objectives and requirements. These decisions cannot be automated directly as they depend on the user's specific desires.

Steps	**Normal**	**Tool**
Selecting a model	Manual	Manual
Selecting a dataset	Manual	Manual
Configuring data transformations	Manual	Manual
Evaluating model performance	Manual	Automated
Defining the compression goal	Manual	Manual
Defining the performance goal	Manual	Manual
Analyzing the deep learning model	Manual	Automated
Identifying the elements to compress	Manual	Automated
Choosing compression techniques	Manual	Automated
Adjusting compression parameters	Manual	Manual
Implementing the compression techniques	Manual	Automated
Applying the compression techniques	Manual	Automated
Fine-tuning or retraining the model	Manual	Automated
Evaluating the compressed model	Manual	Automated
Validating the compression effectiveness	Manual	Automated
Comparing the compressed model with the original model	Manual	Automated

Table 3: Comparison of manual steps and automated steps when compressing a deep learning model

6.3 Overall Performance

This section presents the results of the experiments that evaluated the overall performance of the tool. These experiments were performed by applying the tool to compress different models with different requirement configurations. This section is divided in 4 parts: performance preservation, model size reduction, computations reduction, and inference time reduction.

6.3.1 Performance Preservation

The results presented in this section come from a series of experiments where the focus lies on performance preservation. Every model is evaluated on their respective dataset(s) with three performance reduction thresholds: -1%, -3% and -5% reductions in performance. The tables list for each model-dataset combination the best degree of compression achieved for the given performance threshold, per compression objective.

For the CNN models the results are shown in Table 4, Table 5, and Table 6, for respectively the MNIST, CIFAR-10, and ImageNet datasets. For the Tranformer model, the results are shown in Table 7.

Model	Threshold	Accuracy (%)	Δ Size	Δ Computations	Δ Time
ResNet50	Baseline	94.7	-	-	-
	-1%	93.7	100×	4×	2.2×
	-3%	91.7	100×	13×	4.2×
	-5%	89.7	100×	25×	4.6×
VGG-16	Baseline	93.9	-	-	-
	-1%	92.9	100×	3×	2.8×
	-3%	90.9	100×	8×	3.5×
	-5%	88.9	100×	21×	4.2×
EfficientNet-B4	Baseline	96.1	-	-	-
	-1%	95.1	100×	3×	1.8×
	-3%	93.1	100×	11×	3.5×
	-5%	92.1	100×	15×	4.9×

Table 4: Results of compressing three CNN models with three accuracy constraint levels on the MNIST dataset.

Model	Threshold	Accuracy (%)	Δ Size	Δ Computations	Δ Time
ResNet50	Baseline	94.7	-	-	-
	-1%	93.7	5×	3×	1.9×
	-3%	91.7	9×	5×	2.8×
	-5%	89.7	14×	6×	3.1×
VGG-16	Baseline	93.9	-	-	-
	-1%	92.9	7×	3×	1.6×
	-3%	90.9	11×	5×	2.4×
	-5%	88.9	19×	6×	3.2×
EfficientNet-B4	Baseline	96.1	-	-	-
	-1%	95.1	5×	3×	1.6×
	-3%	93.1	14×	4×	2.5×
	-5%	92.1	17×	6×	3.1×

Table 5: Results of compressing three CNN models with three accuracy constraint levels on the CIFAR-10 dataset.

Model	Threshold	Accuracy (%)	Δ Size	Δ Computations	Δ Time
ResNet50	Baseline	94.7	-	-	-
	-1%	93.7	5×	3×	1.7×
	-3%	91.7	12×	5×	2.2×
	-5%	89.7	17×	7×	3.2×
VGG-16	Baseline	93.9	-	-	-
	-1%	92.9	6×	3×	1.5×
	-3%	90.9	16×	4×	2.6×
	-5%	88.9	20×	8×	3.1×
EfficientNet-B4	Baseline	96.1	-	-	-
	-1%	95.1	5×	2×	1.8×
	-3%	93.1	10×	5×	2.6×
	-5%	92.1	15×	6×	3.5×

Table 6: Results of compressing three CNN models with three accuracy constraint levels on the ImageNet dataset.

Model	Threshold	Accuracy (%)	Δ Size	Δ Computations	Δ Time
BERT	Baseline	94.7	-	-	-
	-1%	93.7	5×	2×	2.2×
	-3%	91.7	12×	4×	3.2×
	-5%	89.7	18×	5×	3.6×

Table 7: Results of compressing BERT with three accuracy constraint levels on the SQuAD dataset.

6.3.2 Model Size Reduction

This section presents the results of the experiments focused on model size constraints. Every model is evaluated on their respective dataset(s) with three reduction targets: -50%, -90% and -99%. For every compressed model the X-fold reduction, the accuracy, the accuracy drop, and the model size are given.

For the CNN models the results are shown in Table 8, Table 9, and Table 10, for respectively the MNIST, CIFAR-10, and ImageNet datasets. For the Tranformer model, the results are shown in Table 11.

Model	Target	Reduction	Accuracy (%)	Δ Acc	Size (MB)
ResNet50	Baseline	-	98.7	0	90.0
	-50%	2×	98.9	0.2	45.0
	-90%	10×	98.9	0.2	22.5
	-99%	100×	98.6	-0.1	0.9
VGG-16	Baseline	-	98.9	0	512.0
	-50%	2×	98.8	-0.1	256.0
	-90%	10×	99.2	0.3	128.0
	-99%	100×	98.9	0	5.1
EfficientNet-B4	Baseline	-	99.1	0	67.7
	-50%	2×	98.9	-0.2	33.9
	-90%	10×	98.6	-0.5	16.9
	-99%	100×	98.4	-0.7	0.7

Table 8: Model Size constraint levels for three CNN models on MNIST.

Model	Target	Reduction	Accuracy (%)	Δ Acc	Size (MB)
ResNet50	Baseline	-	75.2	0	90.0
	-50%	2×	74.6	-0.6	45.0
	-90%	10×	73.9	-1.3	22.5
	-99%	100×	60.0	-15.2	0.9
VGG-16	Baseline	-	74.3	0	512.0
	-50%	2×	74.1	-0.2	256.0
	-90%	10×	72.8	-1.5	128.0
	-99%	100×	63.1	-11.2	5.1
EfficientNet-B4	Baseline	-	80.1	0	67.7
	-50%	2×	79.2	-0.9	33.9
	-90%	10×	78.3	-1.8	16.9
	-99%	100×	61.9	-18.2	0.7

Table 9: Model Size constraint levels for three CNN models on CIFAR-10.

Model	Target	Reduction	Accuracy (%)	Δ Acc	Size MB
ResNet50	Baseline	-	94.5	0	90.0
	-50%	2×	95.3	0.8	45.0
	-90%	10×	92.1	-2.4	22.5
	-99%	100×	67.8	-26.7	0.9
VGG-16	Baseline	-	95.5	0	512.0
	-50%	2×	95.3	-0.2	256.0
	-90%	10×	91.9	-2.6	128.0
	-99%	100×	75.8	-19.7	5.1
EfficientNet-B4	Baseline	-	95.2	0	67.7
	-50%	2×	95.8	0.6	33.9
	-90%	10×	92.7	-2.5	16.9
	-99%	100×	67.7	-27.5	0.7

Table 10: Model Size constraint levels for three CNN models on ImageNet.

Model	Target	Reduction	F1 (%)	Δ F1	Size(MB)
BERT	Baseline	-	74.1	0	450.0
	-50%	2×	73.8	-0.3	225.0
	-90%	10×	71.7	-2.4	112.5
	-99%	100×	54.5	-19.6	4.5

Table 11: Model Size constraint levels for BERT on SQuAD.

6.3.3 Computational Complexity Reduction

This section presents the results of the experiments focused on computations constraints. Every model is evaluated on their respective dataset(s) with three reduction targets: -50%, -75% and -99%. For every compressed model the X-fold reduction, the accuracy, the accuracy drop, and the number of computations are given.

For the CNN models the results are shown in Table 12, Table 13, and Table 14, for respectively the MNIST, CIFAR-10, and ImageNet datasets. For the Tranformer model, the results are shown in Table 15.

Model	Target	Reduction	Acc (%)	Δ Acc (%)	Comp. (GFLOPS)
Resnet-50	Baseline	-	98.7	0	0.4
	-50%	2×	98.5	-0.2	0.2
	-75%	4×	98.3	-0.4	0.1
	-90%	10×	97.7	-1	0.1
VGG-16	Baseline	-	98.9	0	6.7
	-50%	2×	98.7	-0.2	3.4
	-75%	4×	98.3	-0.6	1.7
	-90%	10×	96.8	-2.1	0.6
EfficientNet-B4	Baseline	-	99.1	0	1.5
	-50%	2×	98.7	-0.4	0.8
	-75%	4×	98.5	-0.6	0.4
	-90%	10×	97.4	-1.7	0.2

Table 12: Computations compressions for three CNN models on MNIST.

Model	Target	Reduction	Acc (%)	Δ Acc	Comp. (GFLOPS)
ResNet50	Baseline	-	94.5	0	1.8
	-50%	2×	93.6	-0.9	0.9
	-75%	4×	93.4	-1.1	0.5
	-90%	10×	76.4	-18.1	0.2
VGG-16	Baseline	-	95.5	0	12.2
	-50%	2×	94.8	-0.7	6.1
	-75%	4×	94.1	-1.4	3.1
	-90%	10×	80.2	-15.3	1.2
EfficientNet-B4	Baseline	-	95.2	0	3.3
	-50%	2×	94.3	-0.9	1.7
	-75%	4×	93.6	-1.6	0.8
	-90%	10×	77.7	-17.5	0.3

Table 13: Computations compressions for three CNN models on CIFAR-10.

Model	Target	Reduction	Acc (%)	Δ Acc	Comp. (GFLOPS)
ResNet50	Baseline	-	75.2	0	4.2
	-50%	2×	74.8	-0.4	2.1
	-75%	4×	73.3	-1.9	1.1
	-90%	10×	62	-13.2	0.4
VGG-16	Baseline	-	74.3	0	15.5
	-50%	2×	73.6	-0.7	7.8
	-75%	4×	73.3	-1.8	3.9
	-90%	10×	43.7	-30.6	1.5
EfficientNet-B4	Baseline	-	80.1	0	4.4
	-50%	2×	79.2	-0.9	2.2
	-75%	4×	78.1	-2	1.1
	-90%	10×	62.5	-17.6	0.4

Table 14: Computations compressions for three CNN models on ImageNet.

Model	Target	Reduction	F1 (%)	Δ F1	Comp. (GFLOPS)
BERT	Baseline	-	74.1	0	11.2
	-50%	2×	73.5	-0.6	5.6
	-75%	4×	71.1	-2.6	2.8
	-90%	10×	54.3	-19.8	1.1

Table 15: Computations compressions for BERT on SQuAD.

6.3.4 Inference Time Reduction

This section presents the results of the experiments focused on inference time constraints. Every model is evaluated on their respective dataset(s) with three reduction targets: -25%, -50% and -75%. For every compressed model the X-fold reduction, the accuracy, the accuracy drop, and the inference time are given.

For the CNN models the results are shown in Table 16, Table 17, and Table 18, for respectively the MNIST, CIFAR-10, and ImageNet datasets. For the Tranformer model, the results are shown in Table 19.

Model	Target	Reduction	Accuracy (%)	Δ Acc (%)	Inference Time (ms)
ResNet50	Baseline	-	98.7	0	3.5
	-25%	1.5×	98.6	-0.1	2.6
	-50%	2×	97.9	-0.8	1.8
	-75%	4×	97.5	-1.2	0.9
VGG-16	Baseline	-	98.9	0	6.3
	-25%	1.5×	98.5	-0.4	4.7
	-50%	2×	98.5	-0.4	3.2
	-75%	4×	97.6	-1.3	1.6
EfficientNet-B4	Baseline	-	99.1	0	8.3
	-25%	1.5×	98.8	-0.3	6.2
	-50%	2×	97.8	-1.3	4.2
	-75%	4×	97.7	-1.4	2.1

Table 16: Inference Time compression for three CNN models on MNIST.

Model	Target	Reduction	Accuracy (%)	Δ Acc	Inference Time (ms)
ResNet50	Baseline	-	94.5	0	3.9
	-25%	1.5×	94.1	-0.4	2.9
	-50%	2×	92.4	-2.1	2.0
	-75%	4×	76.4	-18.1	1.0
VGG-16	Baseline	-	95.5	0	6.8
	-25%	1.5×	94.8	-0.7	5.1
	-50%	2×	94.1	-1.4	3.4
	-75%	4×	80.2	-15.3	1.7
EfficientNet-B4	Baseline	-	95.2	0	4.9
	-25%	1.5×	94.3	-0.9	3.7
	-50%	2×	90.6	-4.6	2.5
	-75%	4×	77.7	-17.5	1.2

Table 17: Inference Time compression for three CNN models on CIFAR-10.

Model	Target	Reduction	Accuracy (%)	Δ Acc	Inference Time (ms)
ResNet50	Baseline	-	75.2	0	3.5
	-25%	1.5×	74.5	-0.7	2.6
	-50%	2×	73.1	-2.1	1.8
	-75%	4×	58.3	-16.9	0.9
VGG-16	Baseline	-	74.3	0	6.3
	-25%	1.5×	73.4	-0.9	4.7
	-50%	2×	72.8	-1.5	3.2
	-75%	4×	53.1	-21.2	1.6
EfficientNet-B4	Baseline	-	80.1	0	10.2
	-25%	1.5×	79.2	-0.9	7.7
	-50%	2×	78.1	-2	5.1
	-75%	4×	62.5	-17.6	2.6

Table 18: Inference Time compression for three CNN models on ImageNet.

Model	Target	Reduction	F1 (%)	Δ F1	Inference Time (ms)
BERT	Baseline	-	74.1	0	52.0
	-25%	1.5×	73.5	-0.6	39.0
	-50%	2×	71.1	-2.4	26.0
	-75%	4×	54.3	-19.8	13.0

Table 19: Inference Time compression for BERT on SQuAD.

6.4 Summary

This section presented the results of a series of experiments evaluating the components and performance of the tool. The experiments focused on assessing the successful achievement of the research objectives, and on evaluating the performance of the tool and its underlying modules.

In subsection 6.1 we evaluated the different compression techniques and the effects of combining them. In subsection 6.2 we analyzed what parts of the compression flow got automated and quantified the manual steps that got saved. Finally, subsection 6.3 evaluated the overall performance of the tool. It presents the results of a series of experiments with different models and different compression configurations.

The next section discusses the implications of the results presented in this section.

7 Discussion

This section provides a thorough analysis of the results obtained from the evaluation of EasyCompress, discussing the insights, implications, and potential improvements.

We first discuss the results from the module analysis, where we look into the effects of combining the different compression techniques. We then discuss the results from the automation analysis and the selection system analysis. Next, we analyze the performance of the tool and evaluate how well it can compress the different models towards different compression objectives. We finish this section by discussing the limitations and challenges of the tool and the areas for potential improvement.

7.1 Module Analysis

The results presented in subsection 6.1 show the different effects of applying and combining the different compression techniques. In this section we analyze these results and aim to confirm whether we successfully achieved the objective of combining compression techniques in a complementary way.

First, we presented in subsubsection 6.1.1 the accuracy degradation and recovery graphs of different pruning techniques. There are three observations to be made. The first one is that there is a clear difference in effectiveness of the different techniques, both for degradation and recovery. For degradation the techniques differ in the speed with which the accuracy starts to drop, with some techniques dropping immediately while other techniques are able to maintain the accuracy levels until a critical level of parameters are pruned. Second, we see that despite most pruning techniques showing rapid accuracy degradation when pruning parameters, the potential accuracy recovery remains very strong. When pruning past a certain threshold, performance degrades completely; however, the remaining architecture is still capable of achieving the previous performance levels. Third, we observe the remarkable effectiveness of random pruning. Despite its naive implementation, random pruning appears to preserve the recovery potential quite effectively, closely rivaling the top-performing pruning techniques.

Next, in subsubsection 6.1.2, we compared the effects of the different distillation methods on the performance recovery. We can again observe three things. First, the type of distillation does have an impact on both the speed of recovery and the final recovered accuracy. There is a significant difference between the recovered performance scores by the soft and the hard target distillation. Second, the combined loss distillation seems to be more closely linked to the hard target distillation. This might be due to the influence of the temperature hyperparameter in combined distillation. Third, and most importantly, the distillation leads to higher performance recovery compared to the finetuning recovery. This confirms the complementary benefits of combining pruning and distillation. Introducing distillation allows for better performance recovery than pruning with only finetuning could achieve.

In subsubsection 6.1.3 the effects of the different quantization methods are compared on both uncompressed models, and on models that have been compressed by pruning and distillation. We can observe two things. First there is a small difference in the accuracy degradation when applying quantization. Since all the quantization techniques aim for INT-8 quantization, the main difference comes from the how the quantization map is learned. Second, we observe that the effects of quantization on accuracy drop and size reduction are similar for the uncompressed and the compressed model. This confirms the complementary nature of combining quantization with other compression techniques such as pruning and distillation.

Finally, in subsubsection 6.1.4, we presented a comparative evaluation of the effects of combining the compression techniques. We observe that for the three evaluated models the combination of all three techniques lead to the smallest accuracy reductions. This confirms that the tool combines the compression techniques in an effective manner and that there are complementary benefits of combining techniques.

7.2 Automation Analysis

The automation results presented in subsection 6.2 show that the developed tool successfully accomplished the research objective of building a tool that automates model compression. The tool automates 10 out of the 16 listed manual steps required to compress a deep learning model. However, the steps that require manual intervention are those related to user-specific needs, such as model and dataset selection, and defining the compression objectives. The remaining manual tasks are largely user-dependent, limiting further automation.

The main benefit of the automation algorithm is that it effectively removes repetitive tasks, enabling researchers to focus on strategic decisions. This automation offers significant efficiency gains in working with model compression, yet there is room for future improvements, such as intelligent suggestions for the remaining manual steps. These advancements could further streamline the compression process, increasing the usefulness of the tool during model development and deployment.

7.3 Selection System Analysis

This analysis aims to answer the research question about the influence of the compression goal on the compression action selection. Since the selection system is manually designed, this influence is directly dependent on the design choices made in the development of the tool, which are discussed in subsection 4.2.

The choice of pruning strategy is directly dependent on the compression objective: model size reduction leads to Global or Linear pruning strategies, while computation and inference time reduction leads to Convolutional, Attention, or Global pruning strategies. This was a deliberate design choice, explained in subsection 4.2. The choice of pruning technique depends on a combination of the compression objective and the model architecture. In our experiments the most selected pruning strategy and technique are Global and LAMP pruning. This can be attributed to the architectures of the tested models, and the great general performance of LAMP pruning.

The choice of distillation technique is not influenced by the compression objective, as its main function is to recover the performance of the model. Its actions don't alter the structure of the model and thus can't translate into reductions of any kind. The most selected distillation technique is hard target distillation. This can be attributed to the tested models being evaluated on the CIFAR-10 benchmark, which uses accuracy as a performance metric, for which hard target distillation is optimal.

The choice of quantization technique is not influenced by the compression objective. However, when the compression objective is inference time reduction the quantization technique tends to be omitted. This is because the quantized models can only be run on a CPU backend which is much slower than GPUs. While quantization doesn't directly reduce the number of computations, it does have the capacity to improve the energy efficiency, by optimizing it for CPUs, which are more energy-efficient than GPUs.

The main insight of this analysis is that while the selection of compression actions is influenced by the compression objective, the actual selection in techniques, depends more on other factors, such as model architecture and available compute.

7.4 Performance Analysis

This section analyzes the performance of the tool. We examine how well it can compress towards different objectives with as little performance degradation as possible. We analyze the results from subsection 6.3 and discuss where the strengths and weaknesses lie.

7.4.1 Performance Preservation

When analyzing the performance preservation results of the compressed models we observe four things.

First and foremost, we can confirm that the research objective of developing an automated tool that can compress deep learning models with minimal accuracy loss has been successfully achieved. We find that for each tested model, the tool can reduce both model size and computations at least 2-fold, with a performance-loss less than 1%. Compression for inference time has also been successfully achieved, although to a lesser degree. For most configurations, with different models, datasets, and compression objectives, the reduction are many times higher. With up to 10, 20, and even 100-fold reductions for some configurations.

The second thing that is noticeable is the great performance across all fronts on the MNIST dataset. For every compression level that was tested the performance drop is negligible. This is in contrast with the performance of the tool on the other datasets, where the largest reduction targets lead to significant performance losses. I attribute this difference to the inherent complexity of the given task. The MNIST classification task is fairly simple, while the CIFAR-10, ImageNet, and SQUAD are more complex. For every task there seems to be a minimum required complexity in network architecture to be able to reach a certain performance level successfully. When compression exceeds this threshold, the network's performance begins to degrade.

Third, for the same performance threshold the abilities to compress the model vary significantly depending on the compression objective. The tool is most successful, i.e. has the largest reductions, when compressing for model size, then for number of computations, and lastly for inference time. These differences can be attributed to the natures of the objectives and how they are linked to performance. We dive deeper into this in the next subsections.

Lastly, the tool seems to successfully generalize to different architectures and datasets. The tool was able to compress the models to a significant degree for every dataset. Also for the transformer architecture the tool was successful, however, the compression rates tend to be lower. The tool was first designed and evaluated on CNNs and later added more support for transformers, so I believe extra focus on this architecture category may help improve the scores.

7.4.2 Model Size Reduction

The tool compresses well for model size, with significant reductions: we achieve up to 100-fold reductions for the simplest dataset, MNIST. But also for the other datasets we find significant reductions: 3 to 4 fold reductions in model size with a 1% performance degradation, and up to 10-fold reductions with around 2–3% performance degradation.

As stated previously, for the model size compression objective we observe the largest reductions. When diving deeper we find that the accuracy degradation is highly dependent on the model architectures. We see for example, that for the VGG-16 model the accuracy degradation is less strong compared to the other CNN models. On analyzing this behaviour, we found this is because the selected pruning strategy targeted the model's linear layers. Architectures like ResNet and EfficientNet, which have predominantly convolutional layers, require removing the same proportion of parameters to achieve similar compression rates. However, pruning these layers results in higher reductions in computations and decreased generalizability.

7.4.3 Computational Complexity Reduction

Compression for computational complexity reduction can lead to significant decreases in FLOPS: we find 2-fold reductions over all datasets within 1% of performance degradation. We can achieve more significant reductions, up to 5×, when the performance threshold is lowered to 3% degradation. Higher compression rates, as high as the model size reductions, are not possible without significantly degrading performance.

It is clear that compression for computation reduction has a greater accuracy-degradation per point reduction compared to compression for model size reduction. This difference can be attributed to the differences in influences of parameters. Model size compression is the most general way of compressing, as every parameter has an equal amount of influence on the model size. For computation compression not every parameter has an equal amount of influence on the total number of computations. For example parameters in the convolutional filters of CNNs and in the attention heads of transformers attribute to more computations compared to parameters in a linear layer. Therefore when compressing for model size we can maintain the compute-heavy parameters, which seems to lead to better performance preservation. The fact that for the same reduction multiples the accuracy degrades more for computation compression suggests that the number of computations has a direct effect on the performance.

7.4.4 Inference Time Reduction

When compressing for inference time the tool is also successful: the reductions range from 1.5× with a 1% performance degradation to 2× with a 2% performance degradation. Trying to reduce the inference time more leads to rapid degradation. For the inference time reductions we observe two things.

First, that their reduction multiples are by far the lowest, i.e. for the same permitted accuracy drop they have the smallest reduction of all three objectives. Assuming the same hardware and execution environment, inference time is directly dependent on the number of computations. One could therefore expect the reductions to be near-similar to the reductions in computations. However, this is not the case. This can be attributed to a large fixed-cost in the inference time. To reach an empirical estimation of this fixed-cost, we test this assumption by pruning three models to 99% of computations and model size. The fixed-cost varied per model, but was about 40-50% of the normal inference time without quantization, and 20-30% with quantization.

Second, when conducting the experiments we also notice them to be the flakiest. Inference times may differ significantly between runs. We contribute this to several factors, such as background processes and memory or caching issues. Overall this can be fixed fairly easy by taking multiple measurement.

7.4.5 Comparison

When we compare the performance of EasyCompress to previous research we notice two things.

Method	Model	Δ Size	Δ Accuracy
AMC [22]	ResNet-50	1.8X	-0.8%
EasyCompress	ResNet-50	5X	-1%
QRPK [25]	ResNet-56	11X	-1.2%
PrUE [46]	ResNet-56	54X	-0.9%
AutoCompress [34]	ResNet-18	80X	-0.7%

Table 20: Compression performance comparison of related research on ResNet models and CIFAR-10.

First, while the compression rates of the tool are significant, they are not as high as some of the rates reported by other works such as QRPK [25], PrUE [46] and AutoCompress [34]. I attribute this to three things. First EasyCompress doesn't rely on prior model knowledge so that it can better generalize to different models. Research that does rely on this prior model knowledge can use this to optimally configure and apply proposed techniques. Second, EasyCompress uses structured pruning instead of unstructured pruning. While AutoCompress is able to achieve impressive model size reductions, it needs a dedicated execution environment to capitalize on this reduction. Since our tool is designed to work on standard hardware we relied on structured pruning. Third, EasyCompress currently doesn't use hyperparameter tuning, which the other works do rely on. I believe that introducing hyperparameter tuning could further improve the compression effectiveness, bringing it to the same levels as the reported scores of other research.

Second, while there is research with higher compression rates, the tool does outperform the individual compression techniques that are implemented in the tool, This validates that the combination of different techniques can lead to better compression performance, with a better trade-off between compression and performance.

7.5 Limitations and Challenges

There are a number of limitations and challenges encountered by the tool and this research.

First is the comparative performance towards other compression research. The compression achieved by the tool is significant but is still smaller compared to some of the state-of-the-art techniques. Especially unstructured pruning techniques tend to achieve higher compression rates. As discussed earlier this is a consequence of conscious choice to focus on generalizability. By focusing on techniques that can be generally applied to a wider range of models, we lose the benefits of specialised compression techniques that can achieve higher compression rates.

Second, because the tool applies finetuning through distillation the compression procedure requires a significant amount of time and computational power. Depending on the respective dataset and model a compression procedure can take up from just a few minutes, to a couple of hours. This limits the flexibility and ease of use, since long compression times decrease the speed of development.

Third, the number of experiments was limited. An exhaustive exploration of all different combinations configurations was practically impossible, given the available resources during this research. For this reason the focus of the experiments lied on the overall performance, with less exhaustive experiments on comparing the effects of the different techniques on different modules.

Last, the lack of advanced hyperparameter tuning in the selection system, misses out on potential performance gains. The loss functions, optimizers, and other influential hyperparameters can significantly alter performance of the tool. During development I noticed the influence, but wasn't able to implement a more advanced hyperparameter tuning mechanism due to time constraints.

7.6 Potential Improvements and Future Work

There still remain a number of potential improvements that could be added to improve EasyCompress.

The first and most straightforward improvement is to further build out the set of compression techniques that are offered. Currently the tool offers 5 pruning techniques, 3 pruning strategies, 3 distillation techniques and 3 quantization techniques. These could be further extended, more specific pruning strategies, custom importance criteria for ranking structures to be pruned, advanced distillation techniques that take into account intermediate representations, and mixed-precision quantization for more robust bit-representations.

Second, the selection system can be made more intelligent. This can be done by introducing more specialized pruning strategies that could target more specific layer subsets. Another large improvement to the selection system would be to introduce optimized hyperparameter tuning. There exist a number of different approaches from reinforcement learning to bayesian search, which could improve the selection of compression actions even further.A final improvement I would like to introduce is a reliable performance impact estimation algorithm.

Next, the focus should lie on more generalizability. The main focus of this research was on fully connected and convolutional neural networks, with an extra analysis of transformers. More focus should be put towards transformers as well as other architectures, such as RNNs or GANs.

A final improvement I would like to introduce is a reliable performance impact estimation algorithm. Instead of the user setting a performance threshold and a compression objective and target, the user only sets the compression objective and target. Such an algorithm could then estimate how this compression would affect the performance. It could also be applied in reverse, where a user gives a performance threshold and the algorithm returns an estimation of the achievable compression.

7.7 Summary

This section analyzed the results of our experiments and discussed its implications. It also discussed the tool's limitations and areas for further improvement.

The key finding is that we successfully achieved the objective of creating an automated tool for deep learning model compression. The model is able to achieve compression rates of minimally 2× reductions with less than 1% performance loss for every tested model, with some compressions reaching 10-fold or even 100-fold reductions.

Next to that, we also confirmed the successful achievement of the other research questions and objectives. We confirmed the effectiveness of combining the compression techniques, by analyzing the behaviour of the different techniques, and by verifying that the combinations of compression techniques achieve better compression rates than the individual techniques. We also evaluated the selection system and analyzed how the different compression objectives lead to different compression action combinations. Lastly we also evaluated how the tool has automated most of the manual steps of a normal compression process, only leaving out the steps where manual intervention is desired, such as providing the actual model, dataset, and compression targets.

The implication of the significant reductions in model size, computations, and inference time, without significant reductions in performance is that researchers and developers should incorporate compression as an extra step to their existing deep learning pipelines. In the deep learning community there is currently a pretrain-finetune paradigm, to train large models. Based on the effectiveness of this and other research, I advise to extend this paradigm with a compression step. The new deep learning pipeline should become: pretrain, finetune, and compress.

8 CONCLUSION

This thesis presented and evaluated EasyCompress, an automated tool designed to simplify the deep learning model compression process. EasyCompress alows targeted compression towards three different objectives: reducing model size, inference time, or energy consumption. It is designed with a focus on generalizability towards diverse model architectures and flexibility towards user-defined performance and compression requirements.

A selection system creates a set of tailored compression actions using heuristics-based decision trees that take into account various model characteristics and user-specified requirements. This action set is designed to achieve the compression goal with minimal performance degradation. The available compression actions consist of a diverse range of structured pruning, knowledge distillation and quantization techniques.

An automated implementation algorithm takes the set of compression actions and performs them in a sequential manner. For this it relies on three separate modules each targeting one of the three main compression categories: pruning, distillation, and quantization. The final output is a compressed model that respects the user-defined performance threshold.

The experimental results demonstrate EasyCompress's compression effectiveness on a variety of models. The results demonstrate the tool's capability to reduce model size at least 5-fold, inference time by at least 1.5-fold, and the number of computations by at least 3-fold. Most compression rates are even higher, reaching up to 10, 20, and even 100-fold reductions.

Future enhancements can focus on increasing the set of supported compression techniques, optimizing the tool for more complex and diverse deep learning models, and introducing optimizations for specific hardware architectures.

In conclusion, EasyCompress presents a tool to automatically analyze models and tailor compression actions towards specific real-world compression requirements. Its flexibility to cater to different model architectures and user requirements, along with a user-friendly design, makes it an approachable tool for developers and researchers to explore compression of deep learning models. By facilitating the process of compressing models, the tool encourages the deep learning community to integrate model compression as a permanent and vital part of deploying models to production.

Acknowledgements

I would like to profoundly thank Luis Cruz from TU Delft and Rui Maranhao from UPorto for the valuable supervision, guidance and feedback they provided. I would also like to thank Arie van Deursen and Jan van Gemert for their evaluation of this thesis.

REFERENCES

[1] Nima Aghli and Eraldo Ribeiro. Combining weight pruning and knowledge distillation for cnn compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3191–3198, June 2021.

[2] Han Cai, Yihang Zhang, Xiangyu Zhang, Ji Lin, and Song Han. Structured pruning is all you need for pruning cnns at initialization. *arXiv preprint arXiv:2203.02549*, 2022.

[3] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

[4] Ming Chen and Harbin University of Commerce, China. Applications of deep learning: A review. *Int. J. Comput. Sci. Inf. Technol. Educ.*, 4(2):17–24, November 2019.

[5] Tianyi Chen, Bo Ji, Tianyu Ding, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only train once: A one-shot neural network training and pruning framework. *Advances in Neural Information Processing Systems*, 34:19637–19651, 2021.

[6] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization, 2017.

[7] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Universal deep neural network compression. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):715–726, 2020.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[9] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[10] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Compute and energy consumption trends in deep learning inference, 2021.

[11] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision, 2019.

[12] YingBo Fan, Wei Pang, and ShengLi Lu. Hfpq: deep neural network compression by hardware-friendly pruning-quantization. *Applied Intelligence*, 51(10):7016–7028, Feb 2021.

[13] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning, 2023.

[14] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 69–86. Springer, 2020.

[15] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[16] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. Efficient knowledge distillation from an ensemble of teachers. In *Interspeech*, pages 3697–3701, 2017.

[17] Yuri Gordienko, Yuriy Kochura, Vlad Taran, Nikita Gordienko, Andrii Bugaiov, and Sergii Stirenko. Adaptive iterative pruning for accelerating deep neural networks. In *2019 XIth International Scientific and Practical Conference on Electronics and Information Technologies (ELIT)*, pages 173–178. IEEE, 2019.

[18] Luis Guerra, Bohan Zhuang, Ian Reid, and Tom Drummond. Automatic pruning for quantized neural networks, 2020.

[19] Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[20] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[21] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 28:1135–1143, 2015.

[22] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices, 2019.

[23] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3779–3787, 2019.

[24] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021.

[25] Jangho Kim. Quantization robust pruning with knowledge distillation. *IEEE Access*, 11:26419–26426, 2023.

[26] Jangho Kim, Simyung Chang, and Nojun Kwak. Pqk: Model compression via pruning, quantization, and knowledge distillation, 2021.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[28] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning, 2021.

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

[30] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 673–679. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[31] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global dynamic filter pruning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2425–2432. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[32] Chao Liu, Zhiyong Zhang, and Dong Wang. Pruning deep neural networks by optimal brain damage. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[33] Congcong Liu and Huaming Wu. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156:84–91, 2019.

[34] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4876–4883, 2020.

[35] Xinyu Liu, Baopu Li, Zhen Chen, and Yixuan Yuan. Exploring gradient flow based saliency for DNN model compression. In *Proceedings of the 29th ACM International Conference on Multimedia*. ACM, oct 2021.

[36] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3295–3304, 2019.

[37] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming, 2017.

[38] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference, 2017.

[39] Nilesh Prasad Pandey, Markus Nagel, Mart van Baalen, Yin Huang, Chirag Patel, and Tijmen Blankevoort. A practical mixed precision algorithm for post-training quantization, 2023.

[40] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization, 2018.

[41] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[42] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *CoRR*, abs/1907.10597, 2019.

[43] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. *Deep learning: A Review*, 2018.

[44] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, and Jian Cheng. Bayesian automatic model compression. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):727–736, 2020.

[45] Mingyi Wang, Jianhao Tang, Haoli Zhao, Zhenni Li, and Shengli Xie. Automatic compression of neural network with deep reinforcement learning based on proximal gradient method. *Mathematics*, 11(2):338, 2023.

[46] Shaopu Wang, Xiaojun Chen, Mengzhen Kou, and Jinqiao Shi. Prue: Distilling knowledge from sparse teacher networks, 2022.

[47] Jiaxiang Wu, Yao Zhang, Haoli Bai, Huasong Zhong, Jinlong Hou, Wei Liu, Wenbing Huang, and Junzhou Huang. Pocketflow: An automated framework for compressing and accelerating deep neural networks. 2018.

[48] Linfeng Zhang, Chenglong Bao, and Kaisheng Ma. Self-distillation: Towards efficient and compact neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4388–4403, 2021.

[49] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert, 2020.

[50] Xinyu Zhang, Ian Colbert, Ken Kreutz-Delgado, and Srinjoy Das. Training deep neural networks with joint quantization and pruning of weights and activations, 2021.

[51] Ming Zhao, Meng Li, Sheng-Lung Peng, and Jie Li. A novel deep learning model compression algorithm. *Electronics*, 11(7):1066, 2022.