universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„A Multi-Directional Approach for Accelerating Single-Node Image Classification Neural Network Training via Pruning“

verfasst von / submitted by

Lorenz Kummer BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wilfried Gansterer, M.Sc

Acknowledgements

I want to thank Prof. Wilfried Gansterer for useful advice and supervision, as well as the opportunity to freely pursue my own research interests within this master thesis. I want to thank Prof. Torsten Möller and Samir Moustafa for giving me hardware access necessary to execute large-scale experiments. I want to thank my friends and fellow students Kevin Sidak, Tabea Reichmann, Nikolaus Süß, Christian Luidold, Michael Silber and Thomas Fenz for useful discussions. Last but not least, I want to thank my parents, Christian Klenkhart and Michaela Kummer, as well as my family, particularly my significant other Gabi Babić, who during my years of study endured life with a partner who spent more time with his books and computers than with her, and my daughter Emma Maria Kummer-Babić for their support and being a source of motivation.

I would not have come this far without all of you.

Abstract

Deep neural networks (DNNs) continuously increase in architectural complexity and DNN applications are under pressure to handle an every-increasing amount of data as well as more and more complex problems. This has created demand for approaches reducing time and space complexity of such networks during training and inference, especially for training under resource constraints or inference under time constraints. State-of-the-Art (SotA) works aimed at reducing DNNs' demand for memory and computation time during training and/or the inference can be categorized into techniques applying pruning, quantization, neural architecture search (NAS), distributed learning or a combination of these to the network. In this work, the focus lies on intra-training DNN pruning (i.e., pruning the network during and with the aim of accelerating training) for which new approaches are introduced. This is accomplished by an extensive literature survey and in-depth analysis of the State-of-the-Art in the field of intra training DNN pruning, highlighting current scientific gaps and synthesizing and implementing novel solutions to a selected subset of the identified open research questions. The proposed solutions in this thesis will be subjected to extensive analytical and empirical evaluation in comparison with other State-of-the-Art methods in order to demonstrate their scientific contribution.

Kurzfassung

Die kontinuierliche Zunahme der architektonischen Komplexität von Deep Neural Networks (DNNs) sowie der Druck auf DNN-Anwendungen, eine immer größere Datenmenge sowie immer komplexere Probleme zu bewältigen, hat zu einer Nachfrage nach Ansätzen geführt, welche die zeitliche und räumliche Komplexität solcher Netzwerke während Training und Inferenz reduzieren, insbesondere für Training unter Ressourcenbeschränkungen oder Inferenz unter Zeitbeschränkungen. Arbeiten auf dem Stand der Technik, die darauf abzielen, den DNN-Bedarf an Speicher und Rechenzeit während des Trainings und/oder der Inferenz zu reduzieren, können in Techniken kategorisiert werden, die Pruning, Quantisierung, neuronale Architektursuche (NAS), verteiltes Lernen oder eine Kombination davon anwenden. In dieser Arbeit liegt der Fokus auf dem Intra-Training-DNN-Pruning (d.h. dem Beschneiden des Netzwerks während und mit dem Ziel, das Training zu beschleunigen), für das neue Ansätze vorgestellt werden. Dies wird durch eine umfassende Literaturrecherche und eine eingehende Analyse des Stands der Technik auf dem Gebiet der DNN-Beschneidung während des Trainings erreicht, wobei aktuelle wissenschaftliche Lücken hervorgehoben und neuartige Lösungen für eine ausgewählte Teilmenge der identifizierten offenen Forschungsfragen synthetisiert und implementiert werden. Die in dieser Arbeit vorgeschlagenen Lösungen werden einer umfangreichen analytischen und empirischen Bewertung im Vergleich mit anderen Methoden des Standes der Technik unterzogen, um ihren wissenschaftlichen Beitrag zu demonstrieren.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1. Introduction

With the general trend in machine learning towards larger and more complex models to solve increasingly complex problems more accurately [5], inference in time-critical applications or training under resource and/or productivity constraints is becoming increasingly challenging. These applications include robotics, augmented reality, self-driving vehicles, mobile applications, or scientific research where a high number of trained models is often needed for hyperparameter optimization. Accompanied by this, already some of the most common DNN architectures, like AlexNet by [19] or ResNet by [20], suffer from over-parameterization and overfitting as described by [21, 22]. While the current SotA knows numerous approaches for minimizing the resource consumption of DNNs as outlined in detail in Section 3.4, there is room for improvement in several areas as highlighted in Section 3.4.2, particularly in the area of intra-training DNN pruning. Pruning, for which a more detailed introduction is provided in Section 2.6.5, in the context of this thesis refers to the removal of certain portions of a DNN, e.g. the through the induction of sparsity in its layers weights tensors such that most of their elements are zero. Intra-training DNN pruning has been neglected as a source of training acceleration in favour of accelerating inference due to inference being more often executed than training [2, p. 160] and less complicated to prune since no constraints on heuristic overhead or compatibility with training algorithms exist [7]. Nonetheless, DNN training and its acceleration are deemed crucial by the author for without training, there is no inference and consequentially cheaper neural network training opens the door to more inference applications. The acceleration of training and acceleration of inference through pruning also do not contradict each-other, so ideally an efficient training algorithm outputs a DNN which is capable of efficient inference as well. In Section 3.4.2, several open problems in the field of intra-training DNN pruning are discussed and possible solutions are proposed in Section 4. Particularly Section 4 focuses on how forward and backwards pass can be pruned with distinct heuristics incorporating the unique properties and constraints of these two components of training and how the overhead of specific metrics used in pruning heuristics can be reduced through a more efficient computational strategy. The proposed more efficient computational strategies already lead to previously intractable problems becoming tractable, but is further explored how the incorporation of probabilistic methods can be used to reduce heuristic overhead even more.

1.1. Motivation

In a previous work [23], the author already successfully demonstrated how an information theory motivated heuristic can be used for intra-training quantization. Quantization as

conjectured by the author in [24] however essentially treats the same issue as DNN pruning, NAS or mixed-strategy approaches - the systematic simplification of DNNs overly complex for the tasks they are intended to solve, leading to disproportional computation, memory and storage requirements. Motivated by the successful application of an information theory motivated heuristic in the intra-training quantization domain, the author of this thesis intends to explore - among other avenues of approach - the applicability of an information theory motivated heuristic to intra-training DNN pruning.

1.2. Problem Statement

As outlined in Section 1, there is demand for strategies accelerating DNN training and intra-training pruning is a neglected yet important source of training acceleration. Consequently, it is necessary to document and analyse the SotA in intra-training DNN pruning, to identify scientific gaps which hamper the effectiveness of current SotA methods and to devise novel solutions for accelerating training through pruning that go beyond the SotA.

1.3. Goals

The goals of this thesis are to

- Provide a reader with a background in computer science or mathematics with the necessary preliminary knowledge to understand the reasoning in this thesis.
- Survey the current state in intra-training DNN pruning and provide an extensive overview and analysis of current techniques and trends.
- Based upon this survey, identify current scientific gaps and open research questions.
- For a selected subset of research questions focusing on performance increases during DNN training, synthesize novel solutions, potentially taking into account the authors previous experiences with information theory based intra-training quantization (see Section 1.1).
- Empirically and, where possible, theoretically demonstrate through comparison with competing State-of-the-Art works that the proposed solutions advance the State-of-the-Art in the field of intra training DNN pruning.
- Highlight potential future directions of research.

1.4. Structure and Summary

In Chapter 1 Introduction, the problem, motivation and goals of this work are described on an abstract level. In Chapter 2 Preliminaries, the reader is provided with the necessary background information to understand subsequent chapters and reasoning in this thesis.

Chapter 3 Related Work first describes the methodical approach by which related work was selected, provides a brief overview over historical developments, an extensive survey of contemporary work on the problem, analysis of contemporary work and open research questions. Contemporary approaches that apply pruning during training overwhelmingly (97.78%) introduce advantages in terms of acceleration and compression during the inference phase, 28.88% could produce at least theoretically some advantageous effect during training but do not necessarily document it through experiments or analytical results. Only 14.33% prune during training with the explicitly stated goal of accelerating training. Furthermore and independent of whether an approach is designed to accelerate training or inference, any advantages found by researchers in the domain of intra training pruning is predominantly shown empirically with only 10.5% of the surveyed works including any kind of formal proof, which hampers generalizability. While tendencies towards preferred experiments and metrics exist in the field, this focus on empirical research combined with the fact that no standardized and commonly agreed on set of experiments and metrics exists complicates comparison of different works with one another, forcing researchers to execute large numbers of experiments to fully compare their work to potential competitors. Identified key research questions focus on

- How the computational overhead of pruning heuristics can be lowered, which is crucial given that for any intra-training pruning method to succeed in accelerating training, the incurred overhead must be significantly lower than the speed up of the DNN training process.
- Whether heuristics tailored to the specific needs of forwards- and backwards pass yield any advantage over heuristics that do not treat forwards- and backwards pass pruning separately.
- How to combine structured and unstructured pruning (in unstructured pruning, arbitrary weights can be pruned, while in structured pruning, a certain regularity is incorporated, details see Section 2.6.5) to accelerate training.
- Explore whether an intra training pruning scheme originally targeted at inference acceleration can be adapted for intra training pruning targeted at accelerating training.

.

The Chapter 4 Synthesis derives new approaches from related work for solving the open research questions at hand and discusses the thought process and intuition behind them. Concretely, a new and much more memory efficient method for the computation of Gradient Diversity [25] is introduced, which is then used in a gradient pruning heuristic designed to accelerate the backwards pass through conducting only the most important weight updates. The computational efficiency of this gradient pruning scheme is further improved by the incorporation of a Monte Carlo approach which, during a small predefined number of sampling epochs at the beginning of training, learns the probability with which a certain gradient update is skipped and during later stages of training uses

these probabilities to stochastically decide whether an update is applied, reducing the heuristic overhead by up to two orders of magnitude in the process. To accelerate forward passes during training and potentially during inference as well, a new structured pruning scheme motivated by information theory, particularly Kullback-Leibler Divergence [26], is introduced and combined with unstructured pruning techniques for maximum compression and acceleration. One of the unstructured pruning techniques combined with the novel structured pruning approach using the Alternating Direction Method of Multipliers is based on a modified work originally targeting inference acceleration [18], the other is of the much simpler magnitude pruning type.

In Chapter 5 Performance Model, the performance model used for the evaluation of the approaches derived in Chapter 4 Synthesis is described.

Empirical evaluation is done in Chapter 6 Evaluation: first, an ablation study is conducted and the proposed approach's sensitivity to new hyperparameters is studied on small-scale experiments and finally the worth of the whole approach is demonstrated on large-scale experiments. Using several simple but representative architecture/dataset combinations, the ablation study explores which of the devised novel approaches or combination of approaches for forwards- and backwards-pass acceleration yields the most desirable results in terms of highest acceleration and lowest accuracy loss and generalizes best across different architectures and datasets. The best performing approach, a combination of the novel forwards- and backwards-pass pruning methods with the modified unstructured approach originally targeting inference, is then evaluated with respect to its hyper-parameter sensitivity and compared with several other SotA approaches seeking to accelerate training through pruning on large scale experiments. The large scale experiments show that the novel combined approach reaches or surpasses SotA levels in accuracy degradation in at least 57.33% of the evaluated cases whereby in 46.67% of cases clear superiority is demonstrated and in 14.66% the proposed approach is at least not clearly inferior. In terms of training acceleration, the proposed approach reaches or surpasses State-of-the-Art training acceleration in 75.00% of cases whereby in 58.33% of cases clear superiority is demonstrated and in 16.67% the proposed approach is at least not clearly inferior. With regards to the thesis key research questions, these results indicate that indeed heuristics tailored to the specific needs of forwards- and backwards pass yield a slight advantage over heuristics that do not treat forwards- and backwards pass pruning separately and show that combining structured and unstructured pruning during training has an advantage over approaches that only leverage one of those two types of sparsity. Further, it is shown that an approach originally targeted at inference acceleration can be modified such that it is capable of accelerating training. Concerning the reduction of overhead of the heuristic, it is shown that the synthesized Monte Carlo approach is capable of reducing overhead by up to 2 orders of magnitude while the proposed efficient computation of Gradient Diversity can reduce memory consumption by up to 3 orders of magnitude.

Chapter 7 discusses the contribution of this work and its delimitation from other works in the field. The literature survey included in this thesis, which contributes an extensive listing and categorization of carefully selected SotA works, distinguishes itself from other

surveys in this field through its documentation of experiments and metrics used in the field, which is of value for the entire field because it will enable other researchers in the future to select their experiments in such a fashion that they are comparable with as many competing works as possible. The novel approaches proposed in this thesis contribute through providing answers to the associated research questions and are unique in the pruning heuristics or combination of heuristics they employ as well as the novel techniques used to reduce heuristic overhead by up two two orders of magnitude.

Finally, Chapter 8 Conclusion and Chapter 9 Future Work reflect on the outcome of the newly introduced approaches and discuss possible future directions of research derived from the insights gained in this thesis.

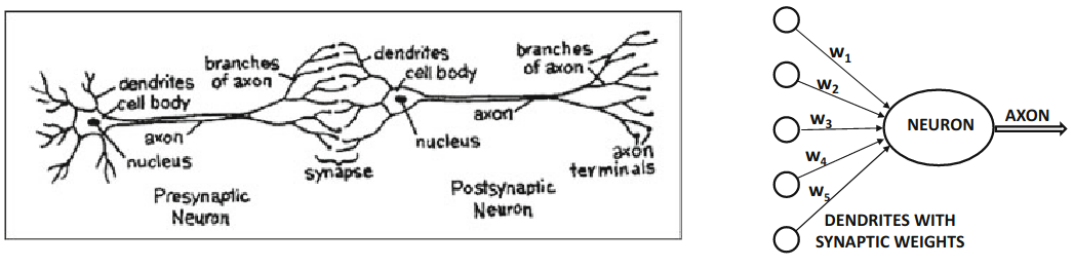
2. Preliminaries

2.1. Notation and Assumptions

In the following, bold printed lower caps variables (e.g., $\boldsymbol{x}$) will be used for column vectors and bold printed upper caps variables (e.g., $\boldsymbol{X}$) will be used for matrices except where otherwise noted. Unless specified otherwise, $\boldsymbol{x}^\top \boldsymbol{y}$ indicates inner product, $\boldsymbol{X}^\top \boldsymbol{Y}$ indicates matrix product and $\boldsymbol{x}^\top \boldsymbol{Y}$ or $\boldsymbol{Y}\boldsymbol{x}$ indicates vector-matrix product of arbitrary matrices $\boldsymbol{X}, \boldsymbol{Y}$ or respectively vectors $\boldsymbol{x}, \boldsymbol{y}$. A variable z denotes a scalar value. The neurons of ANNs will also be referred to as nodes or units synonymously, the connections between neurons will be referred to as connections or weights. Unless specified otherwise, all variables are assumed to be real-valued.

2.2. Biological Motivation and Fundamentals

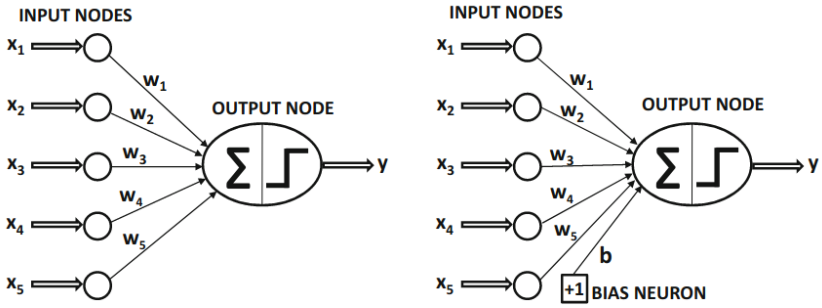
Artificial neural networks (ANNs) are a popular machine learning technique simulating the learning mechanism of certain parts of the human nervous system as depicted in Figure 2.1. In the human brain, neurons are connected to one another via so-called dendrites and axons, whereby the connecting areas between axons and dendrites are referred to as synapses. The strength of these synaptic connections can change frequently dependent on external stimuli [2, p. 1]. In the simplest case of a feed-forward ANN shown



(a) Biological neural network (b) Artificial neural network

Figure 2.1.: Biological and artificial neural network
[2, p. 1]

in Figure 2.2 (a), the single layer Perceptron without bias and scalar output, an input vector $\boldsymbol{x} = (x_1, ..., x_n)$ is multiplied element-wise by a weights vector $\boldsymbol{w} = (w_1, ..., w_n)$ and the aggregated result ($\sum$) is fed into a non-linear activation function (sign function



(a) Perceptron without bias (b) Perceptron with bias

Figure 2.2.: Perceptron
[2, p. 5]

in this exemplary case), i.e.,

$$y = \text{sgn}(\sum_{j=1}^{n} w_j x_j) = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x}) \tag{2.1}$$

Figure 2.2 (b) illustrates the case if an additional bias term b is given s.t. the ANNs output becomes

$$y = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

In the multilayer case (also known as multilayered perceptron) displayed in Figure 2.3 (a)-(d) in various notations, an input vector $\boldsymbol{x}$ is fed to a first layer (the input layer) in the same fashion as in the single layer case. However, the output of the input layer is again used as input for the next layer (a so-called hidden layer) of artificial neurons. This process is repeated recursively for each layer i until a single output y is obtained. Formally, this *forward pass* through the network is denoted as

$$y = \Phi_1(\boldsymbol{w}_1^\top \Phi_2(\boldsymbol{W}_2 \Phi_3(\boldsymbol{W}_3 \boldsymbol{x})))$$

or if bias terms $\boldsymbol{b}_i$ are given

$$y = \Phi_1(\boldsymbol{w}_1^\top \Phi_2(\boldsymbol{W}_2 \Phi_3(\boldsymbol{W}_3 \boldsymbol{x} + \boldsymbol{b}_3) + \boldsymbol{b}_2) + b_1)$$

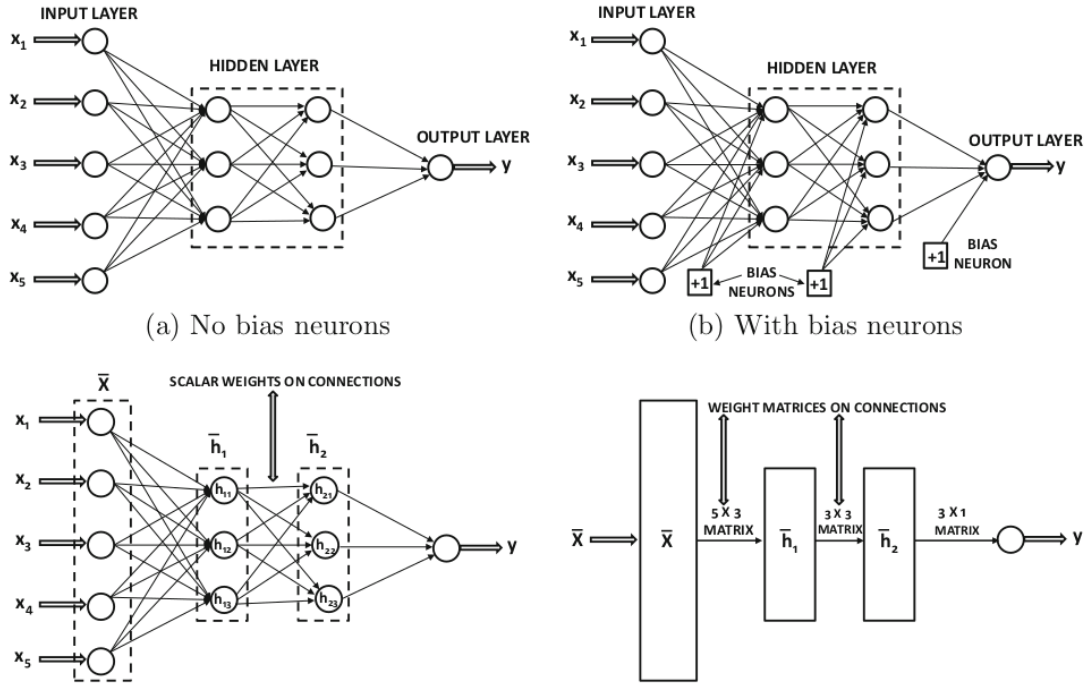
with weights matrices $\boldsymbol{W}_i$ (or vectors $\boldsymbol{w}_i$) and activation functions Φ_i and layer indices $i \in \{1, 2, 3\}$. For multiple outputs as illustrated in Figure 2.4, the output becomes

$$\boldsymbol{x'} = \Phi_1(\boldsymbol{W}_1 \Phi_2(\boldsymbol{W}_2 \Phi_3(\boldsymbol{W}_3 \boldsymbol{x})))$$

or if bias terms $\boldsymbol{b}_i$ are given

$$\boldsymbol{x'} = \Phi_1(\boldsymbol{W}_1 \Phi_2(\boldsymbol{W}_2 \Phi_3(\boldsymbol{W}_3 \boldsymbol{x} + \boldsymbol{b}_3) + \boldsymbol{b}_2) + \boldsymbol{b}_1)$$

Layers following the pattern described in the above simple ANNs are also referred to as



(a) No bias neurons (b) With bias neurons

Figure 2.3.: Multilayer perceptron [2, p. 18]

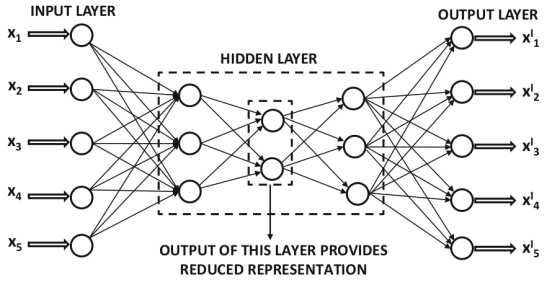


Figure 2.4.: Multilayer perceptron with multiple outputs [2, p. 18]

linear or *fully connected layers*.

For nD inputs (i.e., $\boldsymbol{X} \subset \mathbb{R}^n$) with $n = 2$ such as images, the inputs are either directly reduced to a vector format compatible with linear layers through flattening (usually in row-major order) or through a combination of feature extraction techniques such as, e.g., *convolutional layers* followed by *pooling layers*, which are both briefly covered in Section 2.5.1.2. Because the exact mathematical definitions behind convolutional and pooling layers, their control variables and different subtypes are not directly relevant to this thesis though, no detailed outline is provided here and the interested reader is directed to relevant literature, e.g., [2, p. 315-371].

2.3. Activation Functions

A critical part of ANN design is selecting an activation function, e.g., for the earlier mentioned perceptron, the fact that binary class labels are the prediction target is the main motivation for choosing the sign function in Equation (2.1). Other target variables may require other activation functions. In the case of real values, the *identity* function may be used, or if the probability of a binary class is to be predicted, the *sigmoid* function has the desirable property of continuously mapping the output between 0 and 1. [2, p. 12] The integration of the activation function Φ in a neuron is illustrated in Figure 2.5.

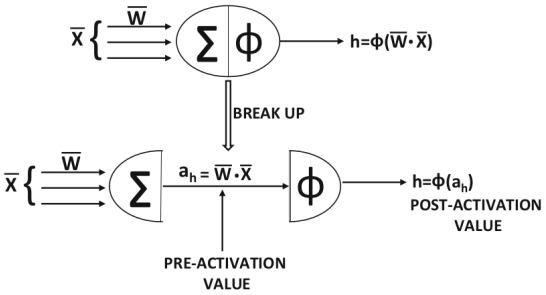


Figure 2.5.: Pre-activation and post-activation values within a neuron [2, p. 12]

2.3.1. Common Activation Functions

While in principle any differentiable function could be used as an activation function, the most common activation functions are briefly discussed below. An illustration of the shape of the below discussed activation functions is given by Figure 2.6.

2.3.1.1. Sign

While the sign activation function allows the mapping to binary outputs at inference time, the fact that it's non-differentiable makes the creation of a loss function for gradient-based training impossible. [2, p. 12]

$$\Phi(v) = \text{sgn}(v) \tag{2.2}$$

2.3.1.2. Sigmoid

Using sigmoid as activation maps to the real interval $0, 1$, which make it useful in cases where the output should be interpreted as probabilities. It is further useful for making probabilistic outputs and creating loss functions based on maximum-likelihood models. [2, p. 12]

$$\Phi(v) = \frac{1}{1 + e^{-v}} \tag{2.3}$$

2.3.1.3. Hyperbolic Tangent (tanh)

The shape of tanh bears similarity to that of the sigmoid function, and tanh is also mathematically related to sigmoid ($tanh(v) = 2 \cdot sigmoid(2v) - 1$). However, it is re-scaled and translated to the interval $[-1, 1]$ and its larger gradient compared to sigmoid improves training. Tanh is preferable over sigmoid if outputs are supposed to be positive and negative. [2, p. 12]

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} \tag{2.4}$$

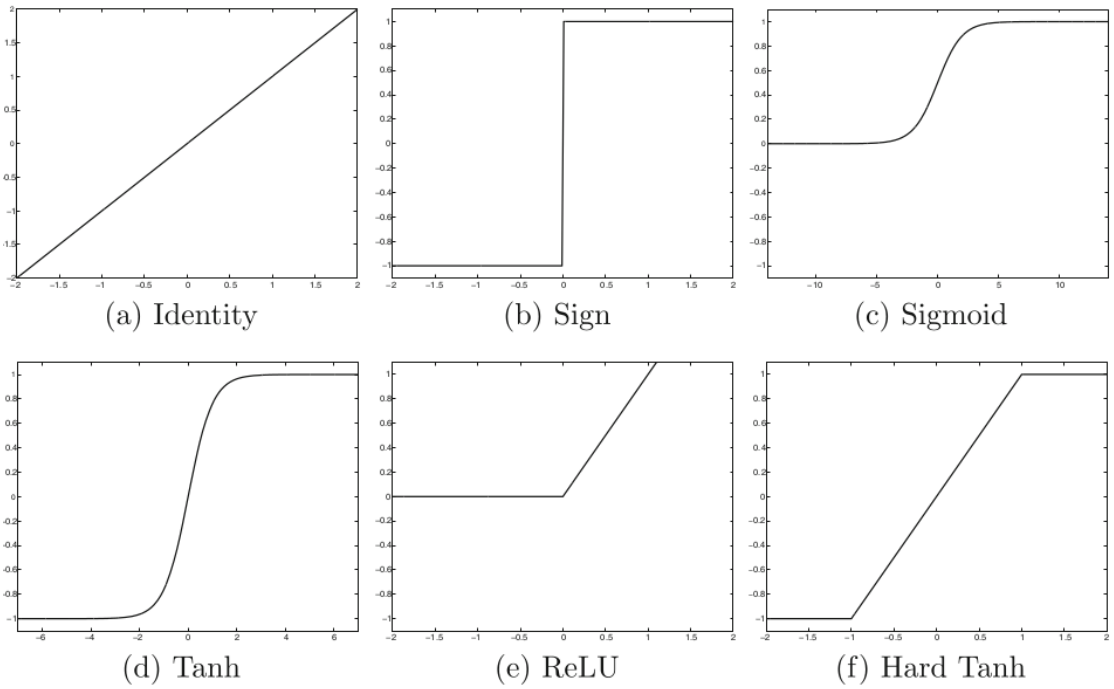


Figure 2.6.: Various activation functions [2, p. 13]

2.3.1.4. Hard tanh

A recent piece-wise linear version of tanh has become popular in recent years and has replaced tanh and sigmoid for easier training in multi layer ANNs. [2, p. 13]

$$\Phi(v) = \max(\min(v, 1), -1) \tag{2.5}$$

2.3.1.5. Rectified Linear Unit (ReLU)

Another piece-wise linear activation function is ReLU, which has become popular for reasons similar to hard tanh. [2, p. 13]

$$\Phi(v) = \max(0, v) \tag{2.6}$$

2.3.1.6. Softmax

In the case of k-way classification (example with $k = 3$ see Figure 2.7), a common choice of activation function for the *output* layer is the softmax function. The softmax function balances the probabilities with respect to C classes. [2, p. 14] For the i-th output, the activation is given by:

$$\Phi(\boldsymbol{v})_i = \frac{e^{v_i}}{-\sum_j^C e^{v_j}} \tag{2.7}$$

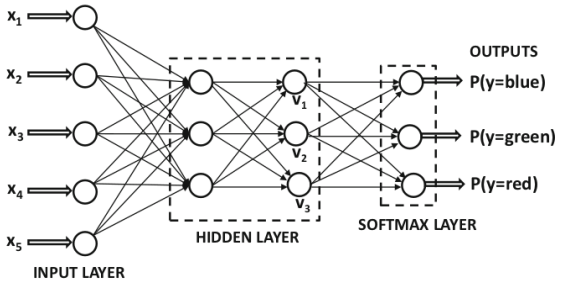


Figure 2.7.: An example of multiple outputs for categorical classification with the use of a softmax [2, p. 14]

2.4. ANN Training and Inference

Generally, the lifecycle of ANNs can be split in two distinct phases: training and inference. During training, the weights of the ANN are adjusted by some optimization algorithm such that a given loss function is minimized. The most common case of ANN training involves a differentiable loss function, to which optionally some regularization terms can be added to reduce the generalization error, and the network is optimized through gradient-based optimization (whereby each optimization step is referred to as*backwards pass*), although solutions for non-differentiable loss functions exist

The training phase is executed only once except in the case of continuous learning. The inference phase, however, can be executed many times over. In the following, I will briefly cover common loss functions, regularization and optimization available in major machine learning frameworks.

2.4.1. Loss Functions

The loss function is an essential component of ANN training, as it provides a real valued representation of the degree to which an ANN's predictions deviate compared to a given ground truth. The chosen loss function should be sensitive to the problem the ANN is intended to solve, e.g., regression with numeric outputs is compatible with a simple squared loss of the form $(y - \hat{y})^2$ for a single training data point whereby ground truth is denoted by y and prediction is denoted by $\hat{y}$ [2, p. 15] [27, p. 181]. Compatible in

this context means that for some problems such as regression, the numerical values are important and the target values must be matched closely to obtain a small error while in other problems such as classification, the exact numerical output is unimportant as long as the class is indicated unambiguously. [28]

2.4.1.1. Cross-Entropy

For discrete multiclass predictions, the softmax output is a common choice, as it can be used to model the probability of a certain input belonging to a certain class. If softmax is used in the output layer, a very popular choice [27, p. 226] for loss function is cross-entropy loss, which is defined as follows [2, p. 118]:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = -\sum_{i}^{C} y_i log(o_i) \tag{2.8}$$

where y_i designates the i-th element of the label vector $\boldsymbol{y}$ and o_i designates the i-th element of the vector $\boldsymbol{o}$ of class probabilities output by the ANN. An alternative notation highlighting the loss function's implicit dependence via $\boldsymbol{o}$ on the network's weights $\boldsymbol{W}$, true-labels $\boldsymbol{Y}$ and input data $\boldsymbol{X}$ is $\mathcal{L}(\boldsymbol{W}; \boldsymbol{Y}, \boldsymbol{X})$. For softmax outputs $\boldsymbol{o}$ and one-hot encoded class-labels $\boldsymbol{y}$, cross entropy is always non-negative and intuitively, it represents the average number of bits needed to encode data coming from a source with distribution P when model Q is used [29]. Hence the lower the value of cross-entropy, the less bits are required to encode data from ground truth's distribution P to model distribution Q, meaning that the model is more accurate.

2.4.1.2. Mean Squared Error (MSE)

For regression problems with multiple data points, MSE, which computes the mean of the squared errors, is a suitable choice. MSE is given by

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{N} \sum_{i}^{N} (y_i - o_i)^2$$

where y_i designates the i-th element of the ground truth vector $\boldsymbol{y}$ and o_i designates the i-th element of the prediction vector $\boldsymbol{o}$. MSE can be interpreted as the cross-entropy between the empirical distribution P and a Gaussian model Q [27, p. 132].

2.4.2. Optimizers

In multilayer ANNs, the loss is a complicated composition function of the weights of the previous layers of the network. The gradient of this function is computed using the backpropagation algorithm which, based on the chain rule from differential calculus, computes the error gradients summation's of local gradient productions over the paths from a node to the output. [2, p. 21, 107-124] After the computation of the gradients, the weights of the ANN are updated. Various different gradient-based weight update strategies exist [2, p. 134-152], two of the most common will be briefly covered below.

2.4.2.1. Stochastic Gradient Descent (SGD)

The basic concept of gradient descent (GD) is to use the gradients computed via chain rule from the loss $\mathcal{L}$ of the ANN over the entire dataset $\boldsymbol{Y}, \boldsymbol{X}$ to update the weights $\boldsymbol{W}^l$ of each of the layers l of the ANN with learning rate η. [2, p. 122]

$$\boldsymbol{W}^l = \boldsymbol{W}^l - \eta \cdot \nabla^l_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W}; \boldsymbol{Y}, \boldsymbol{X}) \tag{2.9}$$

This, however, if often computationally infeasible in modern ANNs with millions of parameters. Because (in most optimization problems) the loss function can be expressed as the linear sum of the losses w.r.t. individual data instances, iteratively updating the weights for individual data instances losses can be shown to ultimately produce the same result as updating them for the loss computed over the entire dataset. [2, p. 122] The update rule for the j-th data $\boldsymbol{y}_j, \boldsymbol{X}_j$ instance is then given by:

$$\boldsymbol{W}^l_{(j+1)} = \boldsymbol{W}^l_j - \eta \cdot \nabla^l_{\boldsymbol{W}_j} \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j) \tag{2.10}$$

The term stochastic in SGD refers to the random order in which the dataset is traversed. The downside of SGD is though that a much larger number of updates has to be computed than is the case in regular SGD, where only a single update per epoch occurs. This is solved by Mini-Batch SGD [2, p. 123], which is a variant of SGD and can be seen as an intermediate between classic GD and SGD in that it updates weights based on the loss of batches of n images instead of single instances or the entire dataset. Usually, any reference in recent literature to SGD is actually to Mini-Batch SGD, so the term will be used in the same manner for the rest of this thesis. Pseudocode for SGD is given by Algorithm. 1.

2.4.2.2. Adam

Adam is another very popular gradient-based weight update scheme operating on mini batches. Unlike SGD, however, it employs a signal-to-noise normalization scheme by inversely scaling the derivative with the exponentially averaged value of j-th parameter $\boldsymbol{W}_j$ and also exponentially smooths the first-order gradient such that momentum is included in the update. Further, it directly addresses the bias inherent in exponential smoothing that occurs when the running estimate of a smoothed value is unrealistically initialized with 0 [2, p. 140]. On computer vision tasks, Adam can be shown to have a faster convergence speed than SGD but suffers from a higher generalization error because SGD is more locally unstable than Adam at sharp minima and can better escape from them to flatter ones [30]. Pseudocode for Adam is given by Algorithm. 2.

2.4.3. Regularization

Regularization is a technique to reduce overfitting, i.e., the model's generalization error. Various different approaches to regularization exist, but the by far most common approach is penalty-term based regularization, where a penalty term is linearly added to the loss

Data: η, $\boldsymbol{W}_0$, $\mathcal{L}$, $\boldsymbol{X}$, $\boldsymbol{Y}$, λ, μ, τ, nesterov
for *j=1 to ...* **do**
  $\boldsymbol{G}_j \leftarrow \nabla_{\boldsymbol{W}_j} \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j)$
  if $\lambda \neq 0$ **then**
    $\boldsymbol{G}_j \leftarrow \boldsymbol{G}_t + \lambda \boldsymbol{W}_{j-1}$
  end
  if $\mu \neq 0$ **then**
    if $j > 1$ **then**
      $\boldsymbol{b}_j \leftarrow \mu \boldsymbol{b}_j + (1 - \tau)\boldsymbol{G}_j$
    else
      $\boldsymbol{b}_j \leftarrow \boldsymbol{G}_j$
    end
    if *nesterov* **then**
      $\boldsymbol{G}_j \leftarrow \boldsymbol{G}_{j-1} + \mu \boldsymbol{b}_j$
    else
      $\boldsymbol{b}_j \leftarrow \boldsymbol{G}_j$
    end
  end
  $\boldsymbol{W}_j \leftarrow \boldsymbol{W}_{j-1} - \eta \boldsymbol{G}_j$
end
Result: $\boldsymbol{W}_t$

Algorithm 1: SGD Pseudo Code adapted from [14], Arguments: learning rate η , network weights at iteration $\boldsymbol{W}_j$, loss function $\boldsymbol{\mathcal{L}}$, input data $\boldsymbol{X}$, labels $\boldsymbol{Y}$, decay λ, momentum μ, dampening τ, nesterov momentum [15]

function $\mathcal{L}$. In order to be compatible with backpropagation training schemes, the penalty term has to be differentiable. Two common penalties used in ANN training are L_2 and L_1 penalties. [2, p. 181]

2.4.3.1. L_2

In L_2 regularization, the added penalty is defined as the sum of the squares of the values of the network's parameters. This penalty drives weights towards zero (although not enforcing exact zero values) with the quadratic term particularly affecting large values which reduces the influence of individual weights on the network output. The strength of the regularization is controlled via a control parameter $\alpha > 0$ which can be tuned via hyperparameter search techniques.

$$\hat{\mathcal{L}}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j) = \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j) + \frac{\alpha}{2} \|\boldsymbol{W}_j\|_2^2 \tag{2.11}$$

Using $L2$ regularization is a rough equivalent to imposing a weight decay after each parameter update and, in terms of a biological interpretation motivated by Section 2.2, can be viewed as a forgetting mechanism which ensures that only repeated updates have

Data: η, β_1, β_2, $\boldsymbol{W}_0$,$\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$,λ, amsgrad
for *j=1 to ...* **do**
  $\boldsymbol{G}_j \leftarrow \nabla_{\boldsymbol{W}_j} \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j)$
  if $\lambda \neq 0$ **then**
    $\boldsymbol{G}_j \leftarrow \boldsymbol{G}_j + \lambda \boldsymbol{W}_{j-1}$
  end
  $\boldsymbol{M}_j \leftarrow \beta_1 \boldsymbol{M}_{j-1} + (1-\beta_1)\boldsymbol{G}_j$
  $\boldsymbol{V}_j \leftarrow \beta_2 \boldsymbol{V}_{j-1} + (1-\beta_2)\boldsymbol{G}_j^2$
  $\widehat{\boldsymbol{M}_j} \leftarrow \frac{\boldsymbol{M}_j}{(1-\beta_1^j)}$
  $\widehat{\boldsymbol{V}_j} \leftarrow \frac{\boldsymbol{V}_j}{(1-\beta_2^j)}$
  if *amsgrad* **then**
    $\widehat{\boldsymbol{V}_j}^{max} \leftarrow max(\widehat{\boldsymbol{V}_j}^{max}, \widehat{\boldsymbol{V}_j})$
    $\boldsymbol{W}_j \leftarrow \boldsymbol{W}_{j-1} - \gamma \frac{\widehat{\boldsymbol{M}_j}}{(\sqrt{\widehat{\boldsymbol{V}_j}^{max}} + \epsilon)}$
  else
    $\boldsymbol{W}_j \leftarrow \boldsymbol{W}_{j-1} - \gamma \frac{\widehat{\boldsymbol{M}_j}}{(\sqrt{\widehat{\boldsymbol{V}_j}} + \epsilon)}$
  end
end
Result: $\boldsymbol{W}_j$

Algorithm 2: Adam Pseudo Code adapted from [16], Arguments: learning rate η, coefficients used for computing running averages of gradient and its square β_1 and β_2, network weights at iteration $\boldsymbol{W}_j$, loss function $\mathcal{L}$, input data $\boldsymbol{X}$, labels $\boldsymbol{Y}$, decay λ, amsgrad variant [17]

a strong effect on the weights, which prevents the model from *memorizing* the training data (i.e., overfitting) such that generalization is improved. Generally, weights tend to move closer to 0 when L_2 regularization is used. [2, p. 183]

2.4.3.2. L_1

In L_1 regularization, the squared penalty used in L_2 regularization is replaced by a penalty on the sum of the absolute magnitudes of the parameters.

$$\hat{\mathcal{L}}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j) = \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j) + \frac{\alpha}{2} \|\boldsymbol{W}_j\|_1 \tag{2.12}$$

A problem that - at least theoretically - arises in L_1 regularization is that the penalty term is non-differentiable for $w_i = 0$, which would make the use of special techniques such as the *subgradient* method necessary to find the partial derivative. In practice, though, the finite precision of computers usually prevents w_i from being *exactly* 0. While L_1 has the same tendency as L_2 to move weights closer to zero, the different way penalization is applied does lead to different properties of L_1 compared to L_2. From an accuracy

point of view, L_2 in most cases outperforms L_1, however L_1 usually leads to more sparse networks. [2, p. 182]

2.4.3.3. Dropout

Instead of applying a penalty term to the loss function, another way to achieve regularization is the random dropping of nodes of a network during each mini batch. If a node is dropped, then all its in- and outgoing connections are dropped as well. The dropping of both in- and outgoing connections is akin to adding noise to a layers in and outputs, which has been shown to have a regularizing effect [31]. The connection dropping also leads to different neural networks being sampled and trained at each mini batch. In order to make predictions with such a trained ensemble of networks for a classification problem, one possibility is to use the complete network and compute the geometric mean of the output nodes, which corresponds to the average of the log-likelihoods. [2, p. 188-189]

2.4.4. Training Visualization

Training is typically illustrated as training, test and validation loss curves (see Figure 2.8) which show the progress (i.e. the minimization of the loss function) over the epochs of training, aggregated as the average loss over all batches of a dataset. Alternatively, it is also common to show training, test and validation error (e.g. the percentage of misclassified images in an image classification scenario) or accuracy (e.g. the percentage of misclassified images). The terms training, test and validation in this context refer to the dataset that is used: the training dataset is used for model training while the test and validation datasets, which are not used for training, are used for model selection (if multiple are trained on the same dataset) in the case of the validation dataset and final model evaluation in the case of the test dataset.

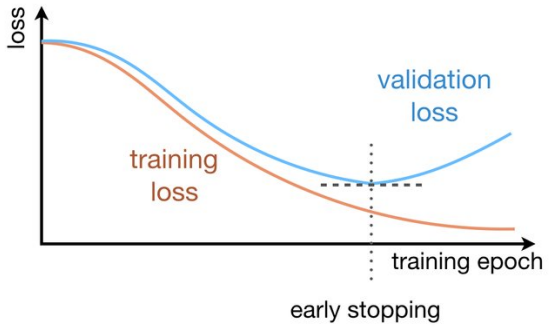


Figure 2.8.: Exemplary illustration of the development of training and validation loss curves during training whereby the models loss is evaluated on the training and validation dataset [3]. In this image, early stopping illustrates the point where further network training will only decrease training loss but increase validation loss due to overfitting [2, p. 27].

2.5. Deep Neural Networks (DNNs)

Survey [1] taxonomizes selected common CNN topologies based on how they obtain representational power: Spatial Exploitation (pixel-neighbourhood correlation, AlexNet[32] LeNet5 [33, 34], VGGNet [35]), Depth (increasing the number of layers to improve target function approximation, InceptionV3 [36] & V4 [37], ResNet152V2[20, 38]), Width (increasing the number of parameters per layer to improve approximation) and Multi-Connection (skipping layers via shortcut-paths, ResNet152V2, MobileNetV1 [39] & V2 [40]). Other categories recognized by [1] are Width-Multi-Connection (Inception family[41, 36, 37]), Feature-Map Exploitation, Channel Exploitation and Attention-based CNNs.

2.5.1. Overview

Due to the large variety of CNN topologies w.r.t. intended task, size and solution strategy, I will focus on a few selected well-known CNNs listed in Table 2.1 intend for image classification using classical reference data sets. Furthermore, my selection is based on exploring CNNs that employ a variety of different strategies (referred to as category, described below in Table 2.1) to improve their results. An (incomplete) taxonomy of CNN topologies is shown in Figure 2.9. It includes most of the selected topologies, which are located in the left half of the taxonomy.

Network	Depth	Parameters	Category	Reference Dataset
LeNet [34]	5	$0.6 \cdot 10^5$	Spatial Exploitation	MNIST [42]
AlexNet [32]	8	$6.0 \cdot 10^7$	Spatial Exploitation	ImageNet [43]
MobileNet [39]	?	$4.2 \cdot 10^6$ (?)	Spatial Exploitation	ImageNet [43]
VGG [20]	19	$1.38 \cdot 10^8$	Spatial Exploitation	ImageNet[43] CIFAR-10 [13]
ResNet [20]	152-110	$1.7 \cdot 10^6$-$2.7 \cdot 10^7$	Multi+Depth	ImageNet[43] CIFAR-10 [13]

Table 2.1.: Overview of the relevant topologies [1]. Depth refers to the number of layers, Parameters to the networks trainable parameters, category to the representational power category, reference dataset to a typical dataset used in conjunction with this topology.

2.5.1.1. Categories

More detailed descriptions of the concepts behind the categories are available at [1].

- Spatial Exploitation-based CNNs: Using the convolution operation as illustrated in Figure 2.10 as feature extraction mechanism [2, p. 315-368], these types of networks exploit the neighbourhood correlation of input data, and different levels of correlation can be explored through using different filter sizes. [1]

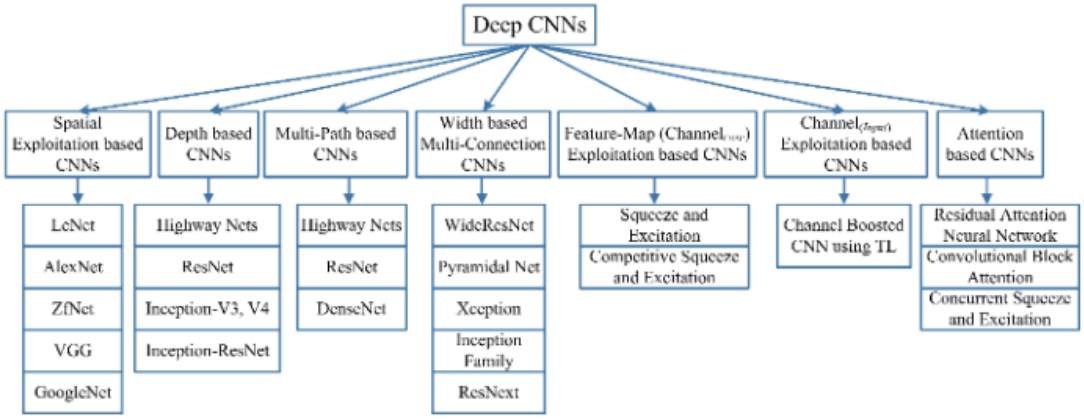


Figure 2.9.: Taxonomy of deep CNN topological categories [1].

- Depth-based CNNs: Networks of this type exploit that with increasing depth, the network becomes better at approximating the target function through a number of non-linear mappings and improved features representations. The main downside of increasing depth in neural networks are the vanishing gradients [2, p. 129-134] problem, as well as negative learning. [1]
- Width-based CNNs: As an answer to the vanishing gradients' problem that might arise as a network's depth (i.e., number of layers) is increased, increasing the width of the network instead was proposed as a viable solution [1].
- Multi-Path based CNNs: In these networks, shortcut paths provide the option of skipping layers deemed less important. Different types of shortcut connections exist in the field, for example zero padded, projection, dropout or 1x1 connections [1].

2.5.1.2. Contemporary Architectures

Below I will briefly introduce contemporary architectures, their motivation, and relevance to contemporary research. The presented architectures were selected based on both their frequency of use in experiments in work related to this thesis (see Table 3.3) as well as their great influence on the development of deep neural networks [1].

LeNet (Spatial Exploitation) LeNet [44] (see Figure 2.11 for topology visualization), a feed-forwards CNN which consists of 5 alternating layers of convolutional and pooling, followed by two fully connected layers, was initially designed to solve the handwritten digit recognition task without being affected by small distortions, rotation, and variation of position and scale. By exploiting the fact that neighbouring pixels are correlated to each other and that such feature motifs are distributed across the entire image, LeNet extracts such features by using convolution with learnable parameters in an effective

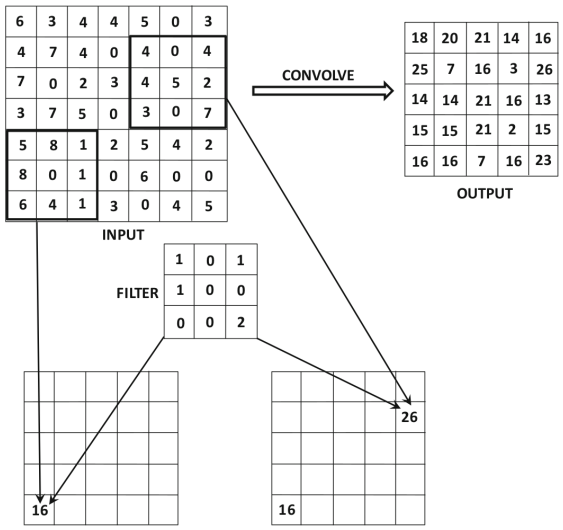


Figure 2.10.: Example of a convolution with a 7x7x1 input and a 3x3x1 kernel with stride 1, illustrating the contribution of each input feature map to the convolutions output. Typically, convolutional layers have several learnable filters and are followed by a pooling layer which aggregates the result obtained by the convolution of each filter with the input. [2, p. 321]

way. [1] LeNet is a small classical CNN topology and, despite its shortcomings (mainly

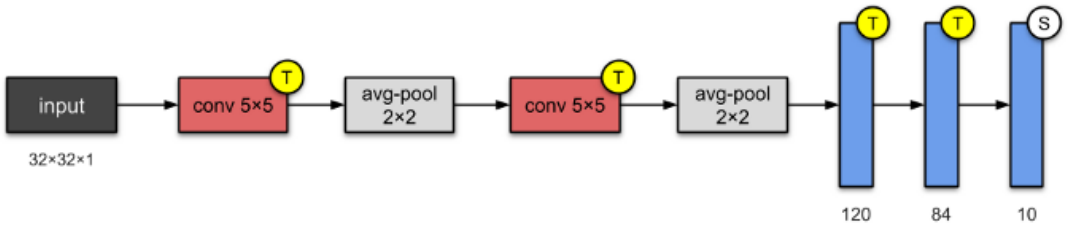


Figure 2.11.: LeNet topology [4]

very poor scaling to diverse classes of images) and age, still relevant until now due to its simplicity and accessibility. LeNet can be trained on MNIST to a high accuracy on most consumer laptops. In the context of contemporary research, it serves as a model for experiments with uncertain outcome because it can be quickly trained and evaluated yet bears sufficient similarity to the more complex architectures.

AlexNet (Spatial Exploitation) To make CNNs applicable to more diverse categories of images, depth was increase from 5 layers in LeNet to 8 layers in AlexNet [19] (see Figure 2.12 for topology visualization). To address the problem that, while increasing

depth improves generalization to different resolutions of images, increasing depth also increases the model's tendency to overfit, AlexNet uses maximum pooling instead of average pooling as well as regularization to learn more robust features as well as ReLU for definition) as non-saturating activation function in order alleviate the problem of vanishing gradients to some degree and to improve convergence rate. Further, AlexNet uses different filter sizes (11x11, 5x5, 3x3) which allows for the extraction of features of variable size. Some of the known disadvantages of AlexNet are that there exist inactive neurons in the first and second layers, as well as aliasing artefacts in the learned feature-maps due to large filter sizes. [1] AlexNet solves many of LeNet's shortcomings is commonly

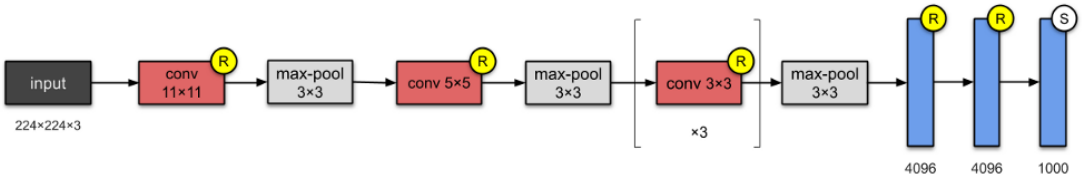


Figure 2.12.: AlexNet topology [4]

used as baseline for new image classification CNN topologies, training, and quantization algorithms [45, 46, 47, 39, 48, 37] or cited as historically relevant by survey papers [1, 49].

MobileNet (Spatial Exploitation) The MobileNet [39] topology was specifically developed to tackle the problem of limited computational resources on mobile devices. It employs a series of 22 convolutional [50] layers, alternating depthwise and pointwise convolutions, in combination called depthwise separable convolutions (Note: a short tutorial on separable convolutions can be found here [51]) to filter and combine inputs into new sets of outputs, followed by average pooling layer, so the topology can be seen as an extreme case in the spatial exploitation category of topologies. The classification is finally done by a single softmax activated linear layer. MobileNet is capable of adapting its width and depth using two hyperparameters α and ρ that control the size of the convolutions and thus the computational complexity of the model. MobileNet is relevant because it provides a topology optimized for use on mobile devices with strong resource constraints without resorting to quantization, i.e., it is a complementary approach to quantization. Furthermore, it has inspired Google Research's powerful Morph Net [52] NAS. A disadvantage of MobileNet is its hyperparameter dependency: α and ρ must be known beforehand, and optimal model size is not determined during training.

Visual Geometry Group (Spatial Exploitation) Visual Geometry Group (VGG) [35] is, at 19 layers deep in the originally proposed configuration, an architecture exploiting depth notably more than AlexNet at 8 layers, but also differs in its arrangement of filters as well as its modularity. VGG uses a large number of, compared to AlexNet, small-sized filters (3x3) placed concurrently to induce the same effect as larger filters at lower computational cost. In VGG, the complexity of the network is further regulated by placing

1x1 convolutions between the convolutional layers, which learn a linear combination of the resultant feature-maps. VGG's main advantages are its simple, homogeneous architecture and increased depth. Its main limitation, however, is that it uses 1.38e8 parameters, which render it computationally complex and deployment on low resource systems difficult. [1] VGG is used as baseline for previously mentioned MobileNet [39] and also referred to in literature [45].

ResNet (Depth+Multi Path) ResNet [20] (see Figure 2.13 for topology visualization), which can be considered a continuation of the trend towards deeper networks, was revolutionary when originally proposed as it for the first time introduced residual learning in CNNs and devised an effective method for the training of such deep networks. The architecture of the residual block of ResNet is illustrated by Figure 2.14. Taxonomically, ResNet falls under the category of Multi-Path CNNs, similar to Highway Networks. ResNet, which in the originally proposed configuration is 20 times deeper than AlexNet and 8 times deeper than VGG, is not only less computationally complex than these architectures, but can in some configurations also achieves higher accuracies on reference image recognition datasets. It further suffers less from the effect of the vanishing gradient problem than other topologies. ResNet's disadvantages are that many layers may contribute very little or no information, the possibility of relearning of redundant feature-maps and highly task-dependent hyperparameters and its comparably complex topology. [1] ResNet is, like

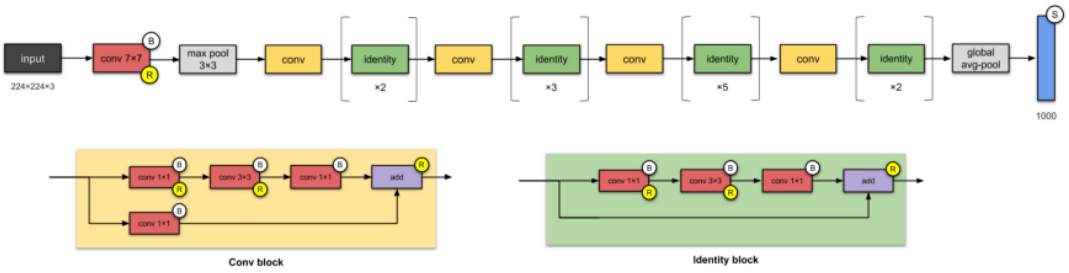


Figure 2.13.: ResNet50 topology [4]

AlexNet, commonly used as baseline for new image classification CNN topologies, training and quantization algorithms [53, 54, 46, 47, 48, 45] or cited as historically relevant by survey papers [1, 49].

2.6. Performance Optimization in DNNs

The growing size and complexity of neural networks led to various performance issues. To measure and compare cost-efficiency of neural networks, accuracy density was devised by [5], which refers to a models test accuracy achieved on a given dataset per the models number of non-zero parameters. As illustrated by Figure 2.15 and 2.16, there exists a general trend that improvements in accuracy come at the price of increased network

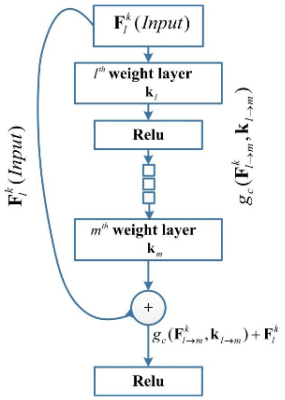


Figure 2.14.: Residual block [1]

complexity and network size, both increasing the burden on the GPU memory 2.17 and reducing throughput 2.18.

As a consequence, various techniques have been developed to either reduce the number of computations, the cost per computation or the memory consumption of the network. While the majority of works are aimed at improving inference performance [2, p. 156-157], there are approaches for improving training or both training and inference as well. In the following, I will briefly introduce four major trends in neural network performance optimization: quantization, neural architecture search, efficient architectures and pruning.

2.6.1. Quantization

One way to accelerate DNN training and/or inference is to choose a more efficient numerical representation of the networks parameters or, within the same representation, reduce the numerical precision used by the network. Numerical representation describes how numbers are stored in memory (illustrated by Figure 2.19) and how arithmetic operations on those numbers are conducted. Commonly available on consumer hardware are floating-point and integer representations, while fixed-point or block-floating-point representations are used in high-performance ASICs or FPGAs. The numerical precision used by a given numerical representation refers to the amounts of bits allocated for the representation of a single number, e.g., a real number stored in float32 refers to floating-point representation in 32-bit precision. With these definitions of numerical representation and precision in mind, most generally speaking, quantization is the concept of running a computation or parts of a computation at reduced numerical precision or a different numerical representation with the intent of reducing computational costs and memory consumption. Quantized execution of a computation, however, typically leads to the introduction of an error, either through the quantized representation's machine epsilon ϵ_{mach} being too large (underflow) to accurately depict resulting real values or the representable range being too small to store the result (overflow).

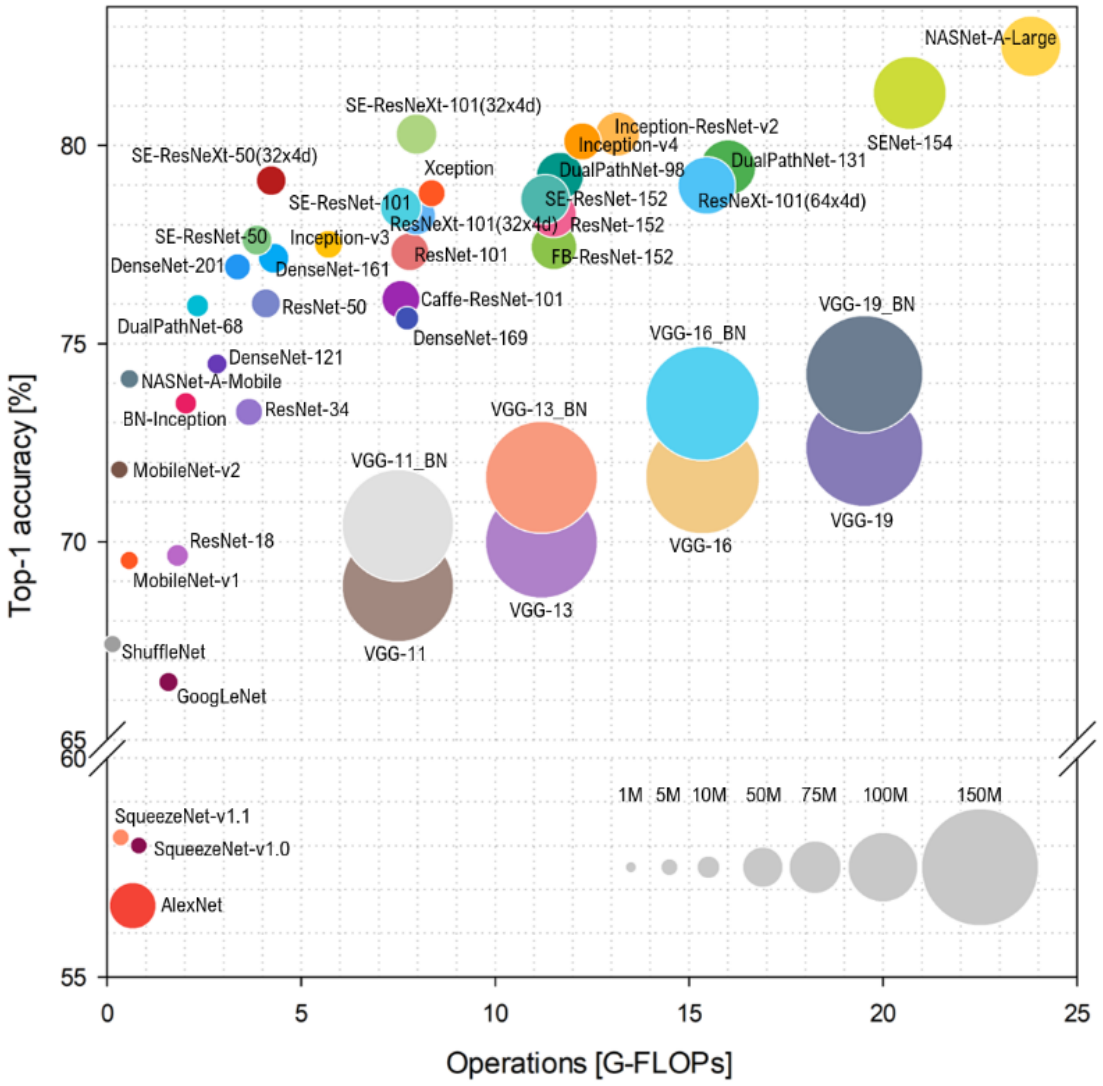


Figure 2.15.: Ball chart showing Top-1 accuracy on Image Net vs. computational complexity. The size of each ball corresponds to the model complexity. [5]

2.6.1.1. Floating-Point Quantization

The value v of a floating-point number is given by $v = \frac{s}{b^{p-1}} \times b^e$ where s is the significand (mantissa), p is the precision (number of digits in s), b is the base and e is the exponent [55, p. 2]. Hence, quantization using floating-point representation can be achieved by reducing the number of bits available for mantissa and exponent, e.g., switching from a float32 to a float16 representation, and is offered out of the box by common machine learning frameworks for post-training quantization i.e. acceleration of inference [56, 57].

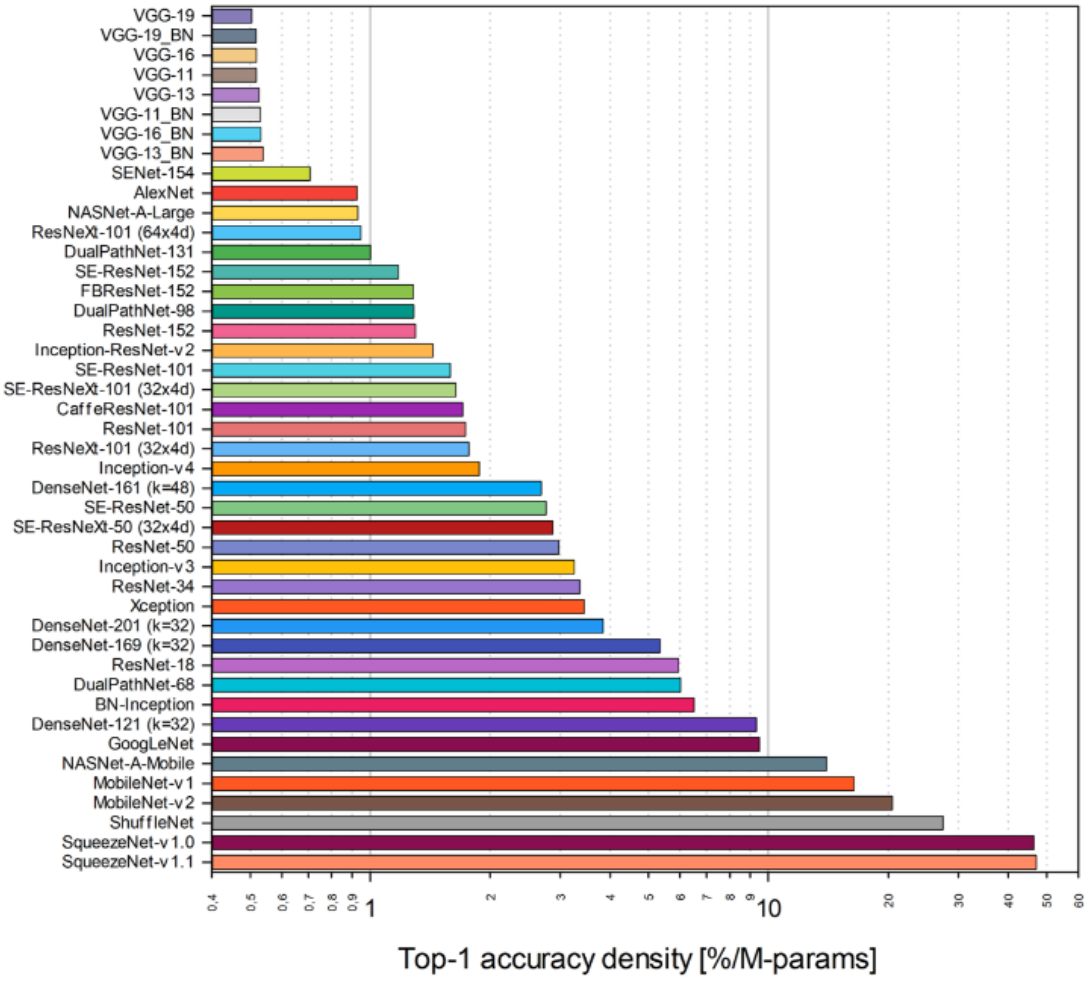


Figure 2.16.: Top-1 accuracy density. The accuracy density measures how efficiently each model uses its parameters. [5]

2.6.1.2. Integer Quantization

Integer representation (int8, int16 due to availability on consumer hardware) is available for post-training quantization and Quantization Aware Training (QAT) [58, 59] in common machine learning frameworks [56, 57]. Quantized training, however, is not supported due to integer quantized activations being not meaningfully differentiable, making standard back-propagation inapplicable [60]. Special cases of integer quantization are 1-bit and 2-bit quantization, which are often referred to as binary and ternary quantization in literature (e.g. [61, 62, 63]).

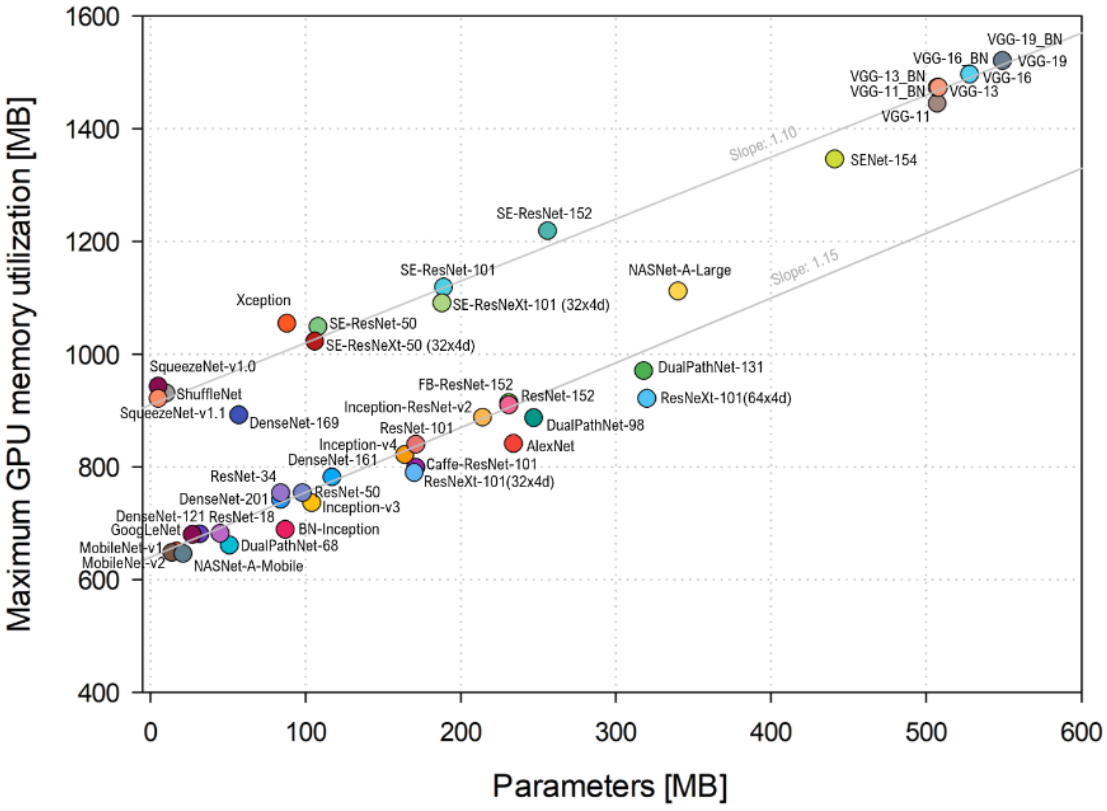


Figure 2.17.: Initial static allocation of the model parameters (i.e., the model complexity) and the total memory utilization. [5]

2.6.1.3. Block-Floating-Point Quantization

Block-floating-point as used by e.g. [64] represents each number as a pair of WL (word length) bit signed integer x and a scale factor s s.t. the value v is represented as $v = x \times b^{-s}$ with base $b = 2$ or $b = 10$. The scaling factor s is shared across multiple variables (blocks), hence the name block-floating point, and is typically determined s.t. the modulus of the largest element is $\in [\frac{1}{b}, 1]$ [55, p. 26-27]. Block-floating-point arithmetic is used in cases where variables cannot be expressed with sufficient accuracy on native fixed-point hardware.

2.6.1.4. Fixed-Point Quantization

Fixed-point numbers have a fixed number of decimal digits assigned, and hence every computation must be framed s.t. the results lies within the given boundaries of the representation [55, p. 1]. By definition of [65], a signed fixed-point numbers of world length $WL = i + s + 1$ can be represented by a 3-tuple $\langle s, i, p \rangle$ where s denotes whether the number is signed, i denotes the number of integer bits and p denotes the number of

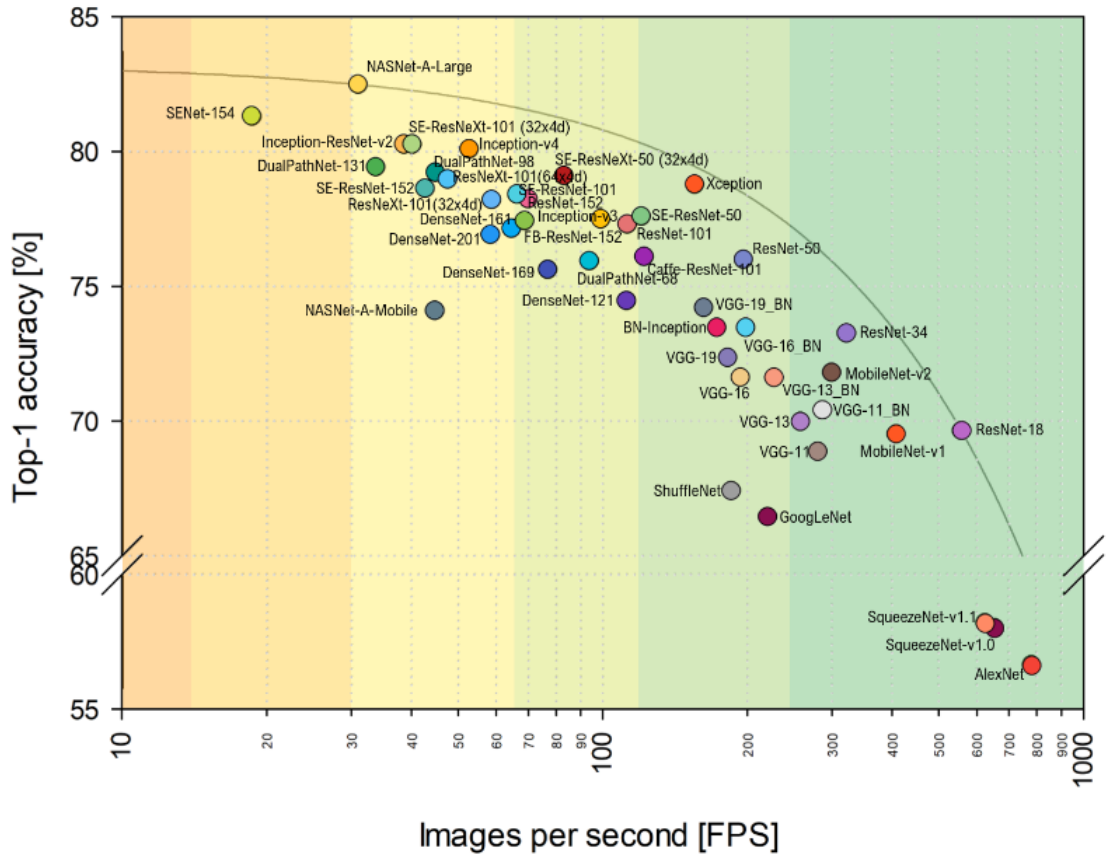


Figure 2.18.: Top-1 accuracy vs. number of images processed per second. [5]

fractional bits. In [23], fixed-point quantization is used in an intra-training quantization scenario.

2.6.2. Distributed Learning

Generally, distributed DNN training involves training the same network architecture instantiated locally on physically independent machines on different batches of data and using some form of synchronization mechanism to update a global model. In Distributed Selective SGD [66], for example, local instances of a DNN model train on locally available data and after each iteration upload the gradients of some selective parameters to the global parameter server. This global parameter server maintains the global parameters, which are determined by the aggregate of all gradients of the local model instances. Next, local model instances download some of the global parameters to update their local parameters. Googles latest distributed training scheme aimed at mobile devices, Federated Training [67], in a similar manner works by a global model being downloaded from the cloud to physically independent machines, trained locally on locally available

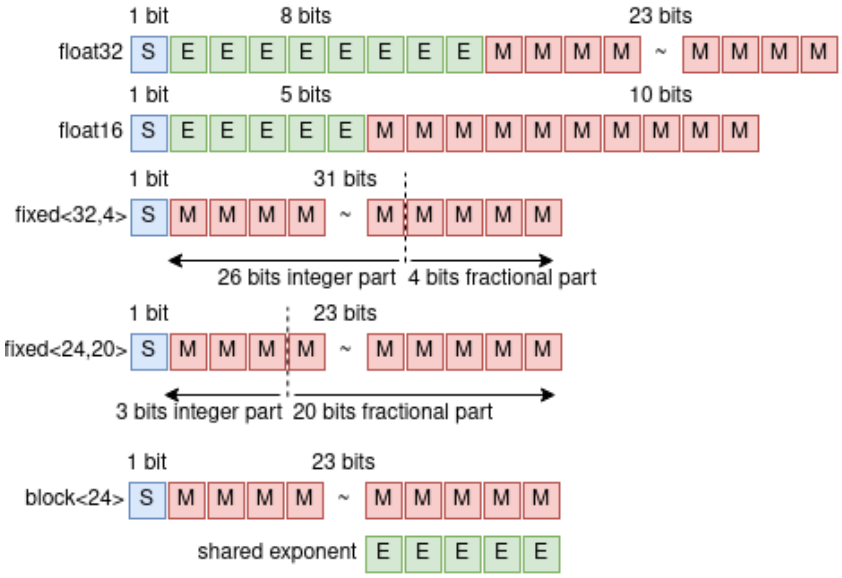


Figure 2.19.: Examples of some possible floating-point, fixed-point and block-floating-point representations at different numerical precisions. Sign (S), Exponent (E), Mantissa (M) bits.

data, and later summarizes the changes to the cloud to improve the global model. To avoid adversarial impact on the mobile device's performance, however, local training only occurs when the mobile device is idle, connected to a power source and connected to a wireless connection. [66]

2.6.3. Neural Architecture Search

Designing neural architectures is a challenging task which traditionally relies heavily on the researcher's prior knowledge and experience, which makes it difficult for users to adapt a given architecture to their needs. Furthermore, the fact that humans tend to think in fixed paradigms and that their decisions are influenced by existing prior knowledge can, to a certain extent, be seen as a limiting factor in the process of scientific discovery. To mitigate these issues, Neural Architecture Search (NAS) was developed: by automating the process of discovering new neural architectures that achieve the best possible performance under certain performance and design constraints [68], reinforcement learning (RL) based methods such as MetaQNN [69] and NAS-RL [70] have demonstrated their capabilities to find architectures reaching SotA accuracy in image classification tasks with minimal human intervention. Other approaches successfully employ evolutionary approaches [71] but come at the costs of very high computational complexity, limiting their usefulness in day-to-day research, which is why large efforts have been devoted to find more efficient [52] approaches to NAS. [72, 73]

2.6.4. Efficient Architectures

A different approach to accelerate ANNs is the introduction of more efficient architectures i.e. architectures that achieve similar test accuracy as a given baseline but have lower memory requirements or computational complexity. Approaches from this category typically employ two different strategies: either new operations for feature extraction are introduced which maintain representational power while being more efficient to compute or classic architectural components are paired with new, compression-oriented components in such a fashion that only a low parameter count in the final model is obtained. A prime representative of the first type is ShuffleNet [48]. In ShuffleNet, two new operations are introduced, pointwise group convolution and channel shuffle, which significantly reduce computation cost while maintaining accuracy. In SqueezeNet [45], which is dedicated to low-end mobile target platforms and a representative of the second category, a new layer called *squeeze layer* is introduced which helps to limit the number of input channels fed to subsequent convolutional layers. SqueezeNet further uses efficient pointwise convolution and employs the strategy of downsampling late in the network so that convolutional layers have large activation maps. The downside of such architectures is that they are static, i.e., they do not dynamically adapt their parameter count to the complexity of the problem. Furthermore these architectures - despite offering a significantly better accuracy/efficiency trade-off than "classic" architectures such as earlier mentioned ResNet or AlexNet, with some authors even observing a 44 times increase in algorithmic efficiency to reach AlexNet level accuracy between 2012 and 2019 [74] - are outperformed in terms of accuracy by the latest more complex approaches such as [75] or [76] as illustrated by Figure 2.15.

2.6.5. Pruning

Neural network pruning, which is also known as sparsity induction or sparsification, refers to the concept of removing certain portions of a neural network's weights (as illustrated by Figure 2.20) or gradients tensors, reducing its size and computational complexity in the process. The sparsity of a tensor is measured as the percentage of zero elements it contains while the opposite concept, density, is measured as the percentage of non-zero elements. This removal of certain parts of weights tensors is, as shown in Section 3, is in most cases implemented by the application of a heuristically-generated binary element-wise multiplicative pruning mask $\boldsymbol{M^l}$ onto a weights (or gradients) tensor $\boldsymbol{W^l}$ of a layer l, zeroing out unwanted elements,

$$\hat{\boldsymbol{W}}^l = \boldsymbol{W}^l \boldsymbol{M}^l \tag{2.13}$$

or by the addition of some penalty term $\mathcal{P}$ to the loss function $\mathcal{L}$

$$\hat{\mathcal{L}}(\boldsymbol{W}; \boldsymbol{y}, \boldsymbol{X}) = \mathcal{L}(\boldsymbol{W}; \boldsymbol{y}, \boldsymbol{X}) + \mathcal{P}(\boldsymbol{W}) \tag{2.14}$$

driving certain areas of the weights tensor $\boldsymbol{W^l}$ towards zero in combination with some magnitude-threshold which sets these near-zero weights to zero. Note that while the first approach can - in principle - be integrated at any point of a neural network's life-cycle (pre- intra- or post-training), the second approach can only be applied during training due

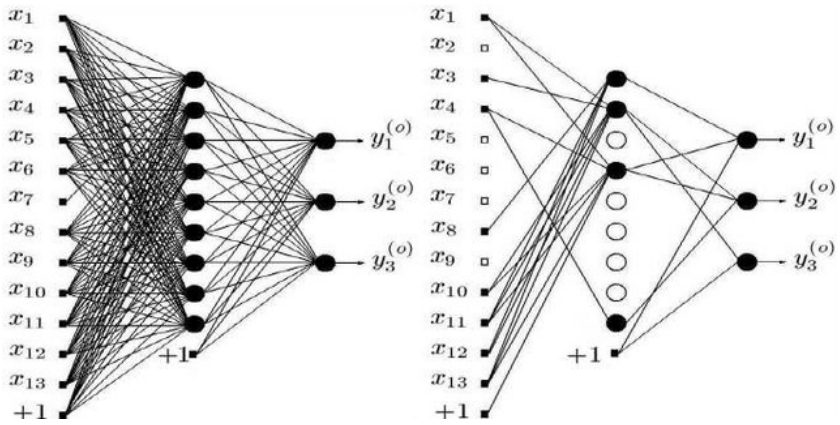


Figure 2.20.: Pruning Weights in a Single Linear Layer [6]

to its dependence on the loss function. Another important distinction is the one between structured and unstructured pruning. Unstructured pruning relies on the pruning of arbitrary weights, leading to irregular, sparse weight matrices, which are less compatible with the data parallel execution model found in e.g. GPUs and multicore CPUs due to the requirement of sparse formats relying on indexing. Structured pruning, on the other hand, seeks to inccorporate a certain type of structure into the weights which can be tailored to match the requirements of the underlying hardware, reducing indexing overhead or, ideally, completely eliminating indexing through maintaining a dense matrix with reduced dimensions. As a consequence, structured pruning has been found to be more hardware friendly in terms of achievable acceleration, particularily on GPUs [77]. Unstructured pruning, however, can be combined with structured pruning to reach higher compression rates [78]. The difference between structured and unstructured sparsity as well as different types of structured sparsity are illustrated in Figure 2.21.

As stated before, the type of structure that is induced has certain implications for the underlying hardware accelerators that are required to be able to leverage a particular type of sparsity. In principle, sparse neural network accelerators can exploit the sparse patterns in weight sparsity, input sparsity, output sparsity, and a combination of multiple patterns. For example, early sparse accelerators as depicted in Figure 2.22 leverage element wise weight sparsity (see Figure 2.21 a)) by storing weights in indexed arrays and naively skipping the multiply–accumulate operations (MACs) with zero values i.e. each processing element (PE) accesses the required activations according to the weight index and then performs the residual MACs. [7] While such an accelerator exploiting weights sparsity can greatly reduce the number of operations and consequently energy consumption and runtime as well, non-zero weights are distributed very irregular, causing large indexing overhead, imbalanced load of PEs as well as inefficient memory access pattern. For this reasons, structured sparsity patterns have been developed which can be more optimally used by underlying hardware. For example, the block-wise sparsity structure illustrated in Figure 2.21 e) and f) can be used to reduce runtime on general-purpose processors [79, 80, 81] by greatly decreasing the required indexes, better balancing

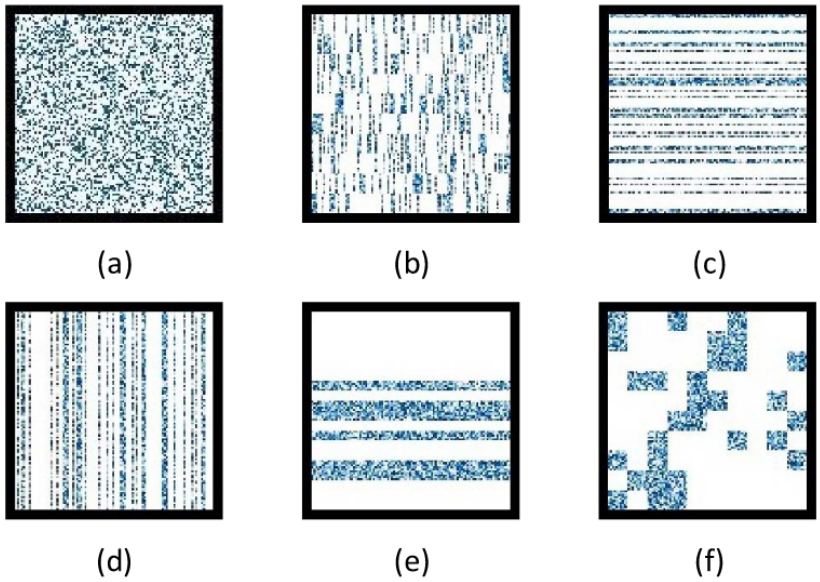


Figure 2.21.: Weight pruning: a) Element-wise, b) – d) Vector-wise, e) and f) Block-wise [7]. White areas indicate zero elements, blue areas indicate non-zero elements.

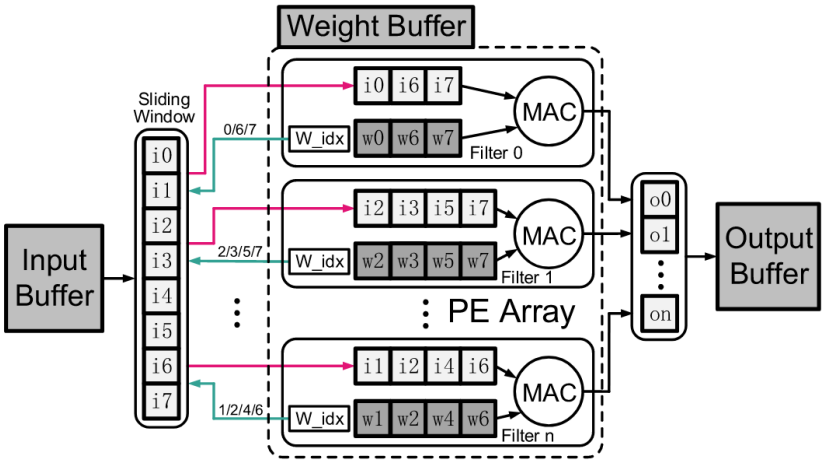


Figure 2.22.: Example of an accelerator [8] with weight sparsity. [7]

of the number of MACs per PE and more efficient memory access. [7] Vector wise sparsity as shown in Figure 2.21 b) - d) can, for example, be exploited by a so called a systolic array [9] as displayed in Figure 2.23. In the systolic array, non-zero weights in multiple columns are combined together by pruning all but the largest elements of each row and a greedy column combining and iterative training is used to guarantee accuracy. In each PE the nonzero weights are buffered with a column index and a multiplexer is used to select

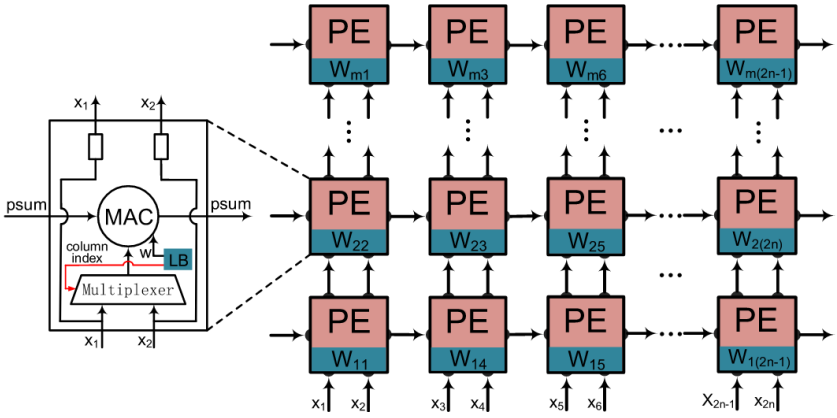


Figure 2.23.: Example systolic array [9] exploiting vector-wise weight sparsity via column combining. [7]

the correct input according to the weights column index. [7] A special type of structured pruning is neuron pruning. In neuron pruning, entire neurons and all their ingoing and outgoing connections are pruned. Generally speaking, neuron pruning often obtains more operation reduction because each neuron is usually connected to many weights, while weights pruning traditionally contributes more to memory savings due to the weights and activations occupying the most memory cost in inference and training. [7] Combining pruning with quantization can create additional sparsity through some values rounding to zero, see Figure 2.24, and also allow to leverage the advantages of quantization mentioned in Section 2.6.1 as well. [7]. Compared to weight pruning, weight quantization is inherently easier to exploit through hardware, because both storage and computation of DNNs are reduced proportionally to the weight precision and no extra overhead due to indexing is incurred [77].

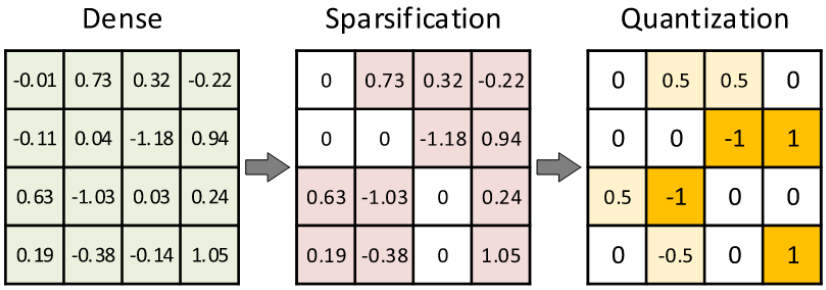


Figure 2.24.: Sparsity can be further increased through quantization [7]

3. Related Work

At the beginning of this chapter, the method for researching and selecting related work is highlighted and a brief but general overview of concurrent techniques for accelerating inference or training of deep neural networks is provided, illustrated by some representatives of each technical category.

For historical development of the field of neural network pruning from 1986 to 2016, first an intentionally wide overview is provided, showing important trends and categories of algorithms related to neural network pruning in general. Particularly influential or interesting works are described in very coarse detail to serve as representatives of their categories. In the end, an analysis of trends some statistics are provided.

The characterization of the State-of-the-Art of intra-training DNN pruning with a particular 2017-2021 focus discusses specific works directly related to the thesis in more detail and provides an in-depth analysis of the relevant papers. As found by [10] and illustrated by Figure 3.1, publication volume on the topic of intra-training pruning has been particularly high during the years 2019 and 2020, which underlines that the selected period is of high interest within the context of this thesis. While the description of historical developments due to its wide scope does not claim completeness in terms of covering each and every related paper published since 1986, the characterization of the State-of-the-Art does uphold this claim within the below described literature research and selection criteria.

3.1. Research and Selection of Literature

3.1.1. Research Method

The papers incorporated in this chapter were researched through the search engines Bielefeld Academic Search Engine (BASE), Connecting Repositories (CORE), Semantic Scholar, Microsoft Academic, Google Scholar, Institute of Electrical and Electronics Engineers (IEEE) Xplore, the ACM Digital Library as well as Neural Information Processing Systems (NeurIPS).

Due to the abundance of works in the field, which is illustrated in Table 3.1, a rigorous selection process had to be applied during the work with academic search engines. Works found via Semantic Scholar, Microsoft Academic and IEEE Xplore were sorted by citation count and the top 50 works found in each search engine were considered for inclusion. Works found via Google Scholar, CORE, BASE, ACM Digital Library, NeurIPS were, due to sorting by citation count being unavailable or delivering irrelevant results (ACM yielded almost exclusively application papers instead of the intended technical papers) on these platforms, ordered by relevancy and the top 50 works were considered for inclusion

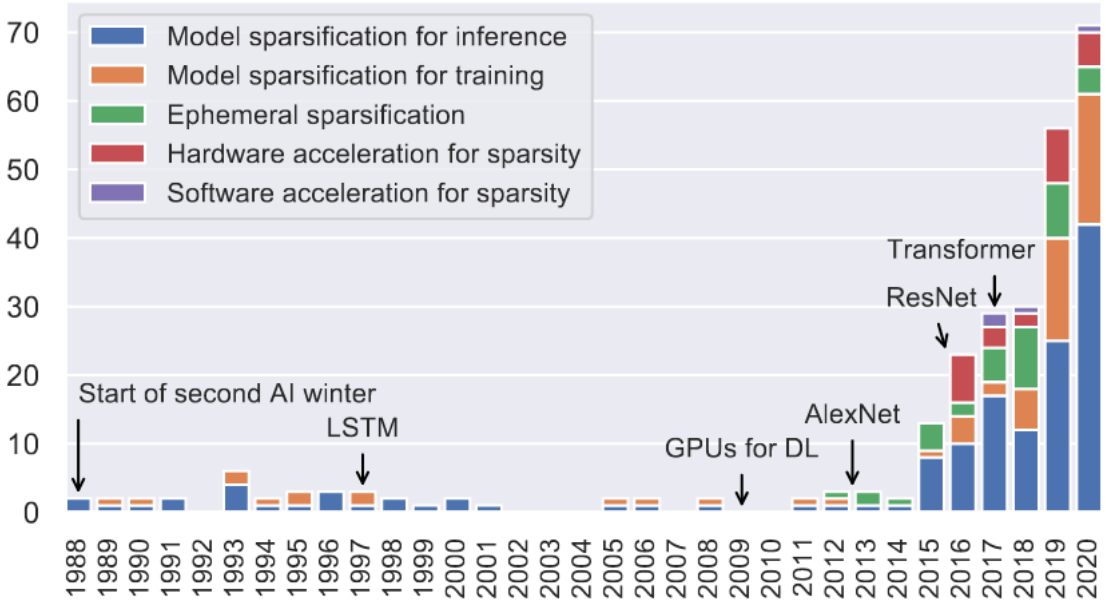


Figure 3.1.: Illustration of the development of the number of scientific publications per year concerning pruning (named sparsification in this image), showing an increased interest in sparse training starting in 2016. The provided labels (LSTM, etc) are of no direct relevance for this thesis and the interested reader is referred to the original publication [10].

if they were published in a journal or conference (Google Scholar, CORE and BASE display arXiv papers that had not been subjected to peer-review as well). ArXiv papers were further only included if deemed particularly influential by citation count and were published in 2021. Furthermore, conference papers were only included if published at a conference with a CORE ranking of A or A* (excellent and flagship conferences) or better and journal papers were only included if they were published in journals ranked Q1 or Q2 quartiles in Web of Science. Where available, it was chosen to limit the search to papers published in English language. Additionally, to the main methodology described in the previous paragraph, papers found in relevant surveys (which themselves were researched as described above) were considered for inclusion as well if they were not already covered by the main methodology.

3.2. Contemporary Directions in Accelerating DNN Training or Inference

Recently published literature on strategies for minimizing DNN training and/or inference runtime and model size while maximizing accuracy can be broadly categorized into the following 4 categories based on the most important methodical concepts:

Search Engine	**Hits**	**Constraint**
Google Scholar	172000	None
Google Scholar	20300	$\geq$ 2017
Google Scholar	13100	Surveys
Google Scholar	6510	Surveys $\geq$ 2017
ACM Digital Library	1023842	ACM Guide to Computing Literature (GCL)
ACM Digital Library	182412	ACM GCL $\geq$ 2017
ACM Digital Library	115094	Research Article in ACM GCL $\geq$ 2017
BASE	4310	None
BASE	2333	$\geq$ 2017
CORE	1833220	English language
CORE	19518	English language $\geq$ 2017
Semantic Scholar	811000	None
Semantic Scholar	371000	$\geq$ 2017
Microsoft Academic	2792	None
Microsoft Academic	1678	$\geq$ 2017
IEEE Xplore	2128	None
IEEE Xplore	1211	$\geq$ 2017
NeurIPS	14	None

Table 3.1.: Search results obtained via different academic search engines on 07.11.2021 for the keywords "neural network pruning" under various constraints. Similar numbers of hits were found for keywords "neural network sparsification", "dnn pruning", "dnn sparsification", "cnn pruning" and "cnn sparsification".

1. Quantization (bit-width reduction of layers, i.e., weights, gradients, activations, fundamentals in Section 2.6.1), to computationally more efficient floating-, fixed- or block-floating-point (an arithmetic approaching floating- on fixed-point hardware) [55] representations [47, 82, 83, 84, 46, 85, 86, 87, 88, 45]. Quantization, in principle, can be applied either globally or on a per-layer basis, pre-training (a priori choice of a certain representation), intra-training (dynamic choice of representation during DNN training) or post-training (after training in float32, quantization to a lower representation is applied).

2. Adapting network architecture or topology (order, type, size or connections of layers, for fundamentals in Section 2.6.3 and Section 2.6.5) either via pruning of over-parameterized weights s.t. layers that do not contribute to the network's accuracy are either reduced in size, bypassed or removed completely and extending weights that under-parameterized [52, 89, 90, 45, 91, 92, 93, 11] or controlling network size via hyperparameters [40, 39, 94, 95, 96]. Similar to quantization, architectural adaption can be applied either globally or on a per-layer basis, pre-training (pruning an architecture before training or re-training it), intra-training (dynamic pruning or extending of the DNNs weights or modifying its topology during training based on

some criterion) or post-training (after training).

3. Introducing architectures obtaining their representational power (representational power based DNN taxonomy see [1], fundamentals in Section 2.6.4) from entirely new, more efficient operations [48, 97, 45]. Approaches employing this strategy usually require training from scratch.

4. Distributed (fundamentals in Section 2.6.2) training and/or inference on large numbers of individually resource constrained devices [49, 67, 98, 66, 85]

Strategies from categories 1 and 2 aim at optimizing DNN parameterization, i.e., reducing over-parameterization either by keeping the number of parameters the same while reducing their numerical representations precision or by reducing the number of parameters while keeping their precision - both approaches lead to reduced model sizes and improved computational efficiency. Approaches falling into categories 3 and 4 do not directly consider the number of model parameters or their numerical representation as relevant variables, but instead focus on more efficient use of computational resources for a given number of parameters and numerical representation. An empirical comparison of the performance gains possible with quantization and pruning can be found in [99].

3.3. Historical Development of DNN Pruning

In order to be able to understand toady's scientific landscape in the field of neural network pruning and to properly contextualize the new approach introduced in this work, it is not only necessary to document the current State-of-the-Art, but also to highlight the historical development of the field. For this reason, this section will describe the path leading to the current State-of-the-Art based on the most influential works of the past.

Finding the optimal topology and architecture has been a topic of interest for researchers in the field since the mid-eighties. Two complementary paradigms of how to achieve this were recognized by authors at the time, but are still relevant to the present-day: constructive or growing approaches and destructive (i.e., pruning) approaches [100, 101, 102, 103, 104, 105]. Constructive approaches start with an underparameterized ANN and add certain architectural or topological elements by a certain heuristic until a certain criterion is satisfied, while destructive approaches assume an overparameterized net is given and prune its architectural or topological elements by a certain heuristic until a certain criterion is satisfied (see Section 2.6.5 for fundamentals). Since this work is dealing with pruning, only related work of the destructive type or where at least some destructive components are present is considered relevant. Purely constructivist methods are not included.

In early neural network pruning literature, several distinctive methodological patterns were documented [100, 106, 101, 105]:

1. Sensitivity methods (Section 3.3.1)

2. Penalty term methods (Section 3.3.2)

3. Other methods (Section 3.3.3)

Works falling into the first category focused on both post- and intra-training pruning while works of the second category almost exclusively targeted network pruning during the training phase. Approaches from the third category cannot be clearly assigned to the first two main categories. Later surveys [107] extended this categorization by

5. Information theory methods

or shifted the focus away from categorizing by pruning methodology to structural properties induced by different pruning techniques [7, 108]. Another categorization found in modern surveys is to differentiate solely between weight, neuron, and layer pruning or to differentiate between by the type of layer that is pruned [109, 110].

6. Structured or unstructured sparsity induction

7. Weight, neuron, filter, or layer pruning

8. Type of layer that is pruned

In addition to these established categories found in the literature, this thesis conjectures the possibility and consequently uses a categorization along whether probabilistic methods are used for pruning. Probabilistic methods, within this context, are understood as methods which either estimate the pruning masks directly or at least estimate some of the heuristics used in the generation of the pruning masks based on some statistical observations. In [111], for example, pruning masks are not exactly computed but instead sampled from a probability distribution which is updated dependent on based on the performance of the model, in [112] the sensitivity score used for generating pruning masks is estimated via variational inference and in [113], the NP-hard problem of using L_0-optimization for pruning is mitigated through using a relaxant probabilistic projection of this norm. Whether this conjectured novel category does have any significance however cannot be conclusively determined within this thesis and is also not part of its goals, hence this task is listed as open research question in Section 3.4.2.13.

In the following, historical related work will be categorized using the methodological categorization approach. For characterizing the State-of-the-Art, the methodological categorization approach (including the conjectured probabilistic methods category) will be used alongside categorization by structural properties (categories 6-8) and information theory methods (category 5, which contains comparably few works) as subcategories which will allow for the best contextualization of this thesis. Another important distinction to be made, particularly for the State-of-the-Art which this work seeks to improve, is whether an approach uses a pre-trained network and prunes it, i.e., only allows gains during inference or is intra-training, i.e., already allows a speed-up and smaller model size during training from scratch. Works that take a pre-trained model, prune it and then re-train it to reduce the pruning-induced error may profit during re-training in terms of computational complexity from pruning but are also not considered to be of intra-training pruning type, see Figure 3.2 for a visualization. While for brevity this distinction is

disregarded for historical developments, the State-of-the-Art will solely be documented in Section 3.4 for true from-scratch intra-training pruning approaches and analysed in Section 3.4.1 only for works seeking to obtain a speed-up or reduce model size during training.

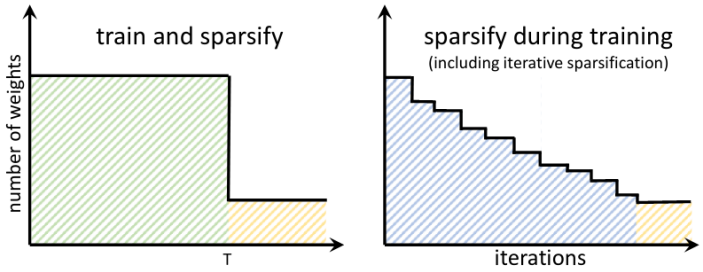


Figure 3.2.: Schematic illustration of the difference between pruning (referred as sparsification in this figure) of a pre-trained network (left) which is pruned at point T and then retrained to regain accuracy and intra-training pruning (right) which starts pruning right at the beginning of training [10]. This thesis is focused on the latter.

3.3.1. Sensitivity Methods

Published in 1989, one of the earliest works on ANN pruning is [114]. The authors introduce a learnable gating term for each unit of an ANN that prunes away connections that are deemed irrelevant to the result of training. Working with a simple 2-layer linear network with 13 trainable parameters in total, in 1990 the authors of [115] demonstrated that using their heuristic based on the sensitivity of the error function w.r.t. individual synaptic connections, they can find those connections causing the least increase in error in pruning. Also in 1990, the influential work Optimal Brain Damage [116] (OBD) introduced the technique of pruning a network dependent on the second order derivatives of the loss function. Using simplifying assumptions, they approximated the prohibitively expensive to compute Hessian by its diagonal elements. Two methods for pruning weights as well as entire neurons based on the frequency and magnitude of excitement of a neuron were proposed in 1993 by [117]. A probabilistic heuristic for selecting pruneable neurons is proposed by [118] but the authors did not empirically demonstrate its capabilities. Based on OBD, in 1993, the first of the highly influential Optimal Brain Surgeon [119, 120] (OBS) series of papers was published (twice, which is why there are two citations). In the initial OBS paper, the authors propose using all second order derivatives (in the form of an efficiently approximated inverse Hessian) of the loss function in a pruning scenario to find weights that can be deleted without causing large increases in error and suggest retraining after pruning. In their 1994 follow-up [121], the authors explore a reduced computational and storage implementation via a dominant eigenspace decomposition and generalize their method to any twice differentiable loss function. OBD and OBS are especially noteworthy due to their methodology being still relevant 26 years later

(see e.g., [89]). A pruning method devised by [122] in 1993 and extended by [123] in 1995 uses statistics gathered during training to estimate whether a certain weight will become zero at some point during training and prunes the network accordingly. A similar approach was taken in 1996 by [124]. Introduced in 1996 by [125] was a sensitivity heuristic based on the product of the gradients of all layers with their corresponding weights for pruning. A 1997 iterative approach by [126] proposes the iterative removal of hidden nodes by some interchangeable sensitivity-based heuristic and then reconstructing the receptive fields (incoming connections) of the subsequent nodes in the network s.t. the overall behaviour of the network is maintained. The authors formulate the receptive field reconstruction problem as a system of linear equations and efficiently solve it using a conjugate gradients least-squares method. In 1999 [127] devised a Kalman filter-based method pruning the weight vector of a single layer ANN in a regression problem. A year 2000 work, [128] repeatedly trains a network by k-fold cross validation, sets the receptive field of a hidden unit to 0 then after retraining removes the hidden unit is removed if the generalization error did not increase. Similarly, irrelevant inputs are removed. A 2005 work proposes pruning hidden neurons based on their statistical average contribution over all inputs seen so far [129] and demonstrates the usefulness of the concept by applying it to a radial basis function (RBF) network. In 2009, [109] uses compressive sampling in a scenario to achieve sparsity in weights matrices and then prune neurons that lost all connections in their projective field. In the same year, [130] showed that the sensitivity of a network's output w.r.t. a hidden node can be computed via Fourier analysis of the variance of the model's response to changes of a node's weight matrix. They employed their approach in an eager fashion. [131] proposes an information entropy-based node pruning heuristic called neural complexity. When pruning, the node causing the least reduction in the network's neural complexity is removed between training steps. A novel sensitivity measure for node pruning in introduced in 2011 by [132] proposes the removal of a node of a layer whose contribution (measured by the sum of its outputs divided by the total number of neurons in the layer) is below a certain threshold. A 2012 work [133] studies the classic OBD frameworks effects on a network's generalization ability, finding that OBD can improve a network's ability to generalize. Instead of following a backwards elimination strategy removing redundant neurons, [134] proposed in 2014 to follow a forward elimination strategy, marking neurons during training that that should stay in the final model dependent on a one-norm based sensitivity metric. Building on a 2015 work [135], [90] in the same year proposed to first train a network, prune its weights by magnitude, retrain the network and then store the remaining sparse weights in a sparse vector format using 8-bits index differences instead of absolute indices to compress the network even more. The network is then further compressed using quantization via weight sharing and Huffman-Coding [136]. [137] in 2016 propose a framework optimizing network structure by selecting significant weights and neurons from a given initial network. Significance is determined using sparse signal representation, which in principle refers to the idea of searching the most compact representation of a signal using vectors in a dictionary [138, 139].

3.3.2. Penalty Terms

In 1988 [140] and later in 1991 [141, 142, 143], works appeared aiming at pruning a network via introducing differentiable penalty terms (e.g., regularization similar as described in Section 2.4.3) into the loss function, driving irrelevant weights to zero during training. Some works proposed removing weights falling below a certain threshold altogether. In 1990, a loss function modified s.t. the number of hidden nodes and the magnitudes of the weights is minimized during training was introduced by [144]. A pruning approach based on incorporating the entropy of the outputs of a layer as a penalty term into the loss function was proposed by [145, 146] in 1996 and extended in 2003, pushing nodes into saturation around 0 or 1 and allowing inactive nodes (i.e., 0) to be pruned. A 2005 pruning solution [147] uses a quadratic penalty term, pushing weights towards 0. After training, it eliminates weights smaller than a certain threshold as well as disconnected nodes. Using zero-norm as penalty term for pruning, [148] in 2012 circumvent the problem that this penalty term is non-differentiable by optimizing the network by Cooperative Swarm training instead of classic gradient-based methods. Published in 2015, [149] proposes to apply two-stage decompositions during training to explore the inter-channel and intra-channel redundancy of convolution kernels. First an initial decomposition based on the reconstruction error of the kernel weights is performed, then the network is fined-tuned under a sparsity constraint by minimizing a sparse group-lasso objective function. Similarly, [150] in 2016 propose a hardware-friendly structured sparsity learning method to directly learn a compressed structure of deep CNNs during the training. They employ a generic regularization based on group-lasso regularization to adaptively adjust multiple structures in DNNs, including structures of filters, channels, intra-layer filter shapes and beyond-layer structure of depth.

3.3.2.1. Weight Decay

Weight decay, which may be seen as a specialized penalty term, as a simple method of exploiting some of the benefits of pruning while avoiding complicating the learning algorithm too much, was first proposed by [151] in 1986. The idea was further developed in 1989 by [152] who introduced a cost function extended by what is today referred to as L_1-regularization [2, p. 182]. A more complex decaying solution was proposed by [153] in 1992, modelling the probability distribution of weights as a mixture of Gaussian's. Unlikely weights have a higher cost under such a distribution s.t. the weights tend to follow the distribution during training.

3.3.3. Other

3.3.3.1. Interactive Pruning

A human in the loop approach was proposed by [154] in 1988 and refined in 1991 by [155]. In these works, a human designer is proposed to inspect a trained network and, based on several heuristics, identifies units that do not contribute to the solution.

3.3.3.2. Bottlenecks

In 1988, a method was described by [156] in which the hidden units compete against each other to survive. The degree to which a layer takes part in the computation depends on the magnitude of its weights, weights that do not contribute to the backpropagation of the error (bottlenecks) are eliminated. In a 1989 extension of that approach, [157] puts certain constraints on weights distance to one another instead of pruning them, leading to a reduction in dimensionality which has an effect comparable to pruning.

3.3.3.3. Genetic and Evolutionary Algorithms

An approach using genetic algorithms for neural network pruning was described by [158] in 1990. In each iteration of their algorithm, a population is composed of differently pruned neural networks and the most successful individuals in terms of weights reduction under the constraint that the network's error has not increased are mated and then retrained. Focusing on recurrent neural networks, [159] in 1994 proposed a network temperature metric as heuristic for choosing the severity of mutations to be performed to each new generation of networks. Weights are mutated through adding Gaussian noise with temperature dependent variance to each connection, nodes are added or removed uniformly, whereby the number of nodes to be removed again depends on temperature. In 1997 [160] presented an evolutionary programming technique, emphasizing the behavioural link between parents and offspring instead of the genetic one. In each iteration of their algorithm, a certain number of the most successful parent networks is selected, mutated to form offspring networks which are then used as parents in the next iteration. Two other 1997 works [161, 162] combine genetic algorithms and simulated annealing to find the best pruned networks. Notable about the second work is its data agnosticism w.r.t pruning superfluous weights and nodes. Inspired by OBS, a 2009 evolutionary data-agnostic pruning algorithm [163] uses Covariance Matrix Adaptation Evolution Strategy [164] to approximate the inverse Hessian efficiently and mutate parameters accordingly. In 2010, [165] proposed particle swarm optimization (PSO) for neural network node pruning. PSO is an evolutionary optimization algorithm searching for a solution through simulating the flocking and movement of birds. Combining an evolutionary algorithm with sensitivity-based pruning and age-based survival selection, [166] in 2012 showed their NAS method is applicable to time-series prediction problems.

3.3.3.4. Vacillating Network Size

In an unusual and interesting approach, in 1997 [167] proposed a training algorithm capable of pruning and expanding a network while training. The authors propose selectively tracking the rate of change of the mean squared error (MSE) of the output layers nodes, extending the number of hidden units for unconverged output nodes while pruning connections or units for converged nodes. [168] proposed in 2010 to apply a coarse-to-fine grained NAS approach. In a first step, a large number of different networks with varying size are created and trained, whereby the size variations are coarse. In a

second step, the best performing network in terms of MSE is chosen and in a fined grained search, minor variations in size are evaluated in the same fashion as used by the coarse grained search.

3.3.4. Analysis of Historical Development

Most striking in the very early neural network pruning literature is the complete lack of a common ground regarding experiments, target devices and metrics in works conducting an empirical evaluation of the developed algorithms, which caused difficulty w.r.t. the comparability of the papers of the time. No "standardized" benchmark architecture/dataset combinations such as AlexNet/CIFAR-10 existed back in the day. Further, architectures were very simple, e.g., ANNs with only a single hidden layer and a few dozen neurons were considered sufficient, and are not comparable to modern architectures with hundreds of layers and millions of neurons. Consequently, porting ideas from these vintage works to modern architectures might prove difficult, although some works such as OBD/OBS [117, 119, 120, 121] stood the test of time and still provide inspiration to modern authors [89]. The most successful methodological concepts in terms of popularity among researchers developed during the historical period from 1986 to 2016 were sensitivity-based (Section 3.3.1) and penalty term (Section 3.3.2) based methods, which are still actively explored today. Research into less successfully concepts, such as interactive pruning (Section 3.3.3.1) and penalty term (Section 3.3.3.2 disappeared over time as they turned out to be overly complex or inefficient and hence inapplicable to all but the tiniest ANNs.

3.4. State-of-the-Art of Intra-Training DNN Pruning

The State-of-the-Art is documented on the following pages. Each work is briefly described in a paragraph and categorized along its fit into the scientific landscape described by categories found in surveys (see above) as well as the experiments conducted by the researchers and metrics used. Metrics are, due to the abundance of different metrics and inconsistent nomenclature in the literature, abstracted and an overview is given in Table 3.2.

MeDNN (2017) [169] is a system for deploying large scale DNNs on mobile devices in a distributed fashion. To leverage the advantages of distributed computation, the authors propose Greedy Two Dimensional Partition (GTDP) to adaptively partition DNN models onto several mobile devices. They further propose Structured Model Compact Deployment (SMCD), an intra-training model pruning scheme targeting the reduction of computing time and communication costs between devices. Training from scratch, SCMD first employs group lasso [170] to learn the structured sparsity of the convolutional layers of a DNN model and then in a refinement phase discards the group lasso penalty while locking the zeroed-out weights to achieve higher accuracy. While aimed at speeding up inference, their method is relevant to this work due to induction of sparsity during

Abstraction	Description
Quality	Degree to which the DNNs task is fulfilled (e.g., accuracy or error)
Latency	Empirical time of DNN, parts of it or communication time
Operations	Count of operations (FLOPs, MACs, cycles)
Instances	Count of weights, channels/filters, neurons, tensor dimensions
Compression	Size reduction (e.g., bytes, number of instances, sparsity)
Acceleration	Reduction of Latency or Operations count (e.g., speed up)
Information	Bit-width used in quantization or bits communicated over network
Energy	Energy consumption (Joule), estimated or empirically
Plasticity	Empirical degree of topological change
Stability	Empirical resilience of DNN to topological change
Memory	Memory consumption, estimated or empirically

Table 3.2.: Abstraction of metrics.

training.
Categories: Penalty term based structured filter-weight pruning
Experiments: CaffeNet (Caffe [171] version of AlexNet [19]) on ImageNet [43]
Targets: Inference
Performance Metrics: Latency, operations, compression, memory
Formal Proofs: None

DEEP R (2017) Motivated by recent discoveries in neurobiology [172, 173, 174, 175], DEEP R [176] is a sparsity inducing training scheme that automatically rewires the DNN s.t. connections are there where they are needed most to solve a given task. At the same time, their total number is strictly bounded. Theoretically, DEEP R views rewiring as stochastic sampling of DNN configurations from a posterior that combines the data likelihood and a specific connectivity prior in a Bayes optimal manner.
Categories: Probabilistic unstructured weight pruning
Experiments: LeNet-300-100 or LeNet-5 [44] on MNIST [42], custom CNN on CIFAR-10 [13], custom LSTM [177] on TIMIT [178] dataset.
Targets: Inference
Performance Metrics: Quality, compression
Formal Proofs: Convergence

Sparse Variational Dropout (2017) Variational Dropout [179] is an extension of Gaussian dropout [180], which, together with binary [181] dropout, is a regularizer commonly used to reduce overfitting in overparameterized DNNs [182, 181]. However, instead of injecting noise, a model can be regularized by reducing the number of its parameters. Extending Variational Dropout further, [183] proposes a new approximation of the Kullbeck-Leibler divergence [26] term in Variational Dropouts objective function as well as a method to reduce the variance of the stochastic gradient estimator, leading to faster

convergence and a sparsified DNN.
Categories: Sensitivity based probabilistic unstructured weight pruning
Experiments: LeNet, VGG [35] on CIFAR-10, CIFAR-100 [13]
Targets: Training and inference
Performance Metrics: Quality, compression
Formal Proofs: None

Input Layer Regularization (2017) By applying $L1$, which under certain conditions produces sparse solutions equivalent to those of L_0 [184, 185] or dedicated sparsity inducing $L_{1/2}$ [186] regularization to the input layer, the authors aim at eliminating redundant dimensions in the network's input layer s.t. redundant nodes can be removed [187].
Categories: Penalty term based unstructured pruning of (input layer) weights
Experiments: 3-layer linear NN on Monk, Sonar, MNIST datasets
Targets: Inference
Performance Metrics: Quality, instances
Formal Proofs: None

Scalpel (2017) The authors of Scalpel [81] observed that in many cases, sparsity can hurt network performance and encoding the sparse format of pruned networks can lead to increased storage space requirements. They overcame this through Scalpel, which customizes DNN pruning to the underlying hardware by matching the pruned DNN's structure to the underlying data-parallel hardware. Their pruning mechanism either maintains weights in fixed-size groups to fully utilize SIMD units on microcontrollers, prunes nodes instead of weights for optimal performance on high-parallelism hardware such as GPUs without sacrificing the dense format or a combination of the previous two for moderate-parallelism hardware such as CPUs. While Scalpel is not a true intra training approach because it employs a train-prune-retrain scheme of operation, i.e., it works on pre-trained DNNs, it is still included here because of its valuable insight that traditional pruning approaches can hurt performance and pruning should be customized to underlying hardware.
Categories: Sensitivity based structured weight and/or node pruning with hardware constraints
Experiments: LeNet-300-100, LeNet-5 on MNIST, ConvNet, NIN on CIFAR-10, AlexNet on ImageNet
Targets: Inference
Performance Metrics: Quality, compression, acceleration
Formal Proofs: None

Bayesian Log-Normal Multiplicative Noise (2017) Going in a similar direction as above-mentioned Sparse Variational Dropout [183] by extending traditional dropout techniques [179, 180, 181], the authors of this paper propose a new dropout-like regularizer capable of inducing structured sparsity [188]. They add a certain multiplicative noise to layer outputs but put a sparsity-inducing log-uniform prior over the noise variables

instead of the weights, thus inducing group-wise sparsity.
Categories: Sensitivity based probabilistic structured weight pruning
Experiments: LeNet-5, LeNet-500-300 and LeNet-5-Caffe on MNIST, VGG on CIFAR-10
Targets: Inference
Performance Metrics: Quality, operations, acceleration, Instances
Formal Proofs: None

Group Sparse Regularization (2017) [189] simultaneously optimizes the weights of a DNN, the number of neurons of each layer as well as the subset of active input features. This is accomplished by extending the group lasso s.t. group level sparsity is enforced on the network's connections. Each group is defined as the set of weights outgoing from a node. These weights can be related to an input node, a hidden neuron or a bias unit dependent on the specific case.
Categories: Penalty term based unstructured (input)-weights pruning
Experiments: Custom MLP on SSD, MNIST and COVER datasets
Targets: Inference
Performance Metrics: Quality, compression, instances
Formal Proofs: None

Global Supervised Low-rank Expansion and Adaptive Drop-weight (2017) (GSLRE, ADW) are two techniques devised in [190] for compressing and speeding up CNNs in an image recognition task. While GSLRE decomposes convolutional layer filters into two low-rank filters, ADW prunes weights by larger than a dynamically adapted threshold, which itself is determined by a sensitivity heuristic.
Categories: Sensitivity based unstructured weight pruning
Experiments: Custom CNN on CASIA-HWDB [191]
Targets: Inference
Performance Metrics: Quality, instances, compression, operations
Formal Proofs: None

A Systematic DNN Weight Pruning Framework using ADMM (2018) was proposed by [18]. The DNN weight pruning problem is formulated as a non-convex optimization problem which is reformulated and solved via the alternating direction method of multipliers (ADMM) [192] with sparsity requirements specified as combinatorial constraints. Given the formulation of the pruning problem is non-convex, optimality of the solution cannot be proven. After a certain number of iterations of ADMM training, the remaining non-zero weights are retrained without constraints using regular SGD.
Categories: Penalty term based structured weights pruning
Experiments: LeNet-5, LeNet-300-100 on MNIST, AlexNet on ImageNet
Targets: Inference
Performance Metrics: Quality, instances, compression
Formal Proofs: Convergence

Feature Boosting and Suppression (2018) (FBS) [193] proposes reducing the computational cost by predictively amplifying salient channels or suppressing unimportant channels in convolutional layers at runtime. Their method does not permanently remove networks structures, but instead the augmented DNN learns the sensitivity of weights w.r.t. to layer inputs during training, allowing dynamic input-dependent pruning at runtime. Layers other than convolutional layers are not affected by their method.
Categories: Sensitivity based structured channel pruning
Experiments: VGG-16, ResNet-18 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations, acceleration, memory
Formal Proofs: None

Rethinking the Smaller-Norm-Less-informative-Assumption (2018) A widely used assumption states that parameters or features with a smaller norm play are less crucial for inference. The authors of [194] propose a channel pruning scheme for accelerating CNNs that does not depend on this assumption. In an end-to-end stochastic training method, they first force the outputs of some channels to be constant almost everywhere by an adaption of batch normalization [195] (BN). Then they these channels are pruned by adjusting the biases of affected layers s.t. the resulting compressed model can be fined-tuned within a few iterations. While network training itself is conducted with regular SGD, the degree to which channels are pruned (i.e., the level of sparsity achieved) is optimized using Iterative Shrinkage-Thresholding Algorithm [196](ISTA).
Categories: Penalty term based structured channel pruning
Experiments: ConvNet and ResNet-20 [20] on CIFAR-10, ResNet-101 on ImageNet, DenseNet [197]/Inception [41] based custom CNN on MSRA10K [198], DUT-Omron [199], Adobe Flickr-portrait [200], Adobe Flickr-hp [200], COCO-person [201]
Targets: Inference
Performance Metrics: Quality, instances, compression, operations
Formal Proofs: None

Single Shot Network Pruning based on Connection Sensitivity (2018) (SNIP) [202] introduces a saliency criterion based on connection sensitivity to identify structurally important connections in the network and prunes the network prior to training which is conducted in a standard manner (SGD, Adam). They achieve this by separating the weight of a connection from whether the connection is present through the introduction of an auxiliary variable which is used to measure the effect removing the weight has on the loss function and optimizing both problems individually.
Categories: Sensitivity based unstructured weight pruning
Experiments: LeNet-5, LeNet-300-100 on MNIST, AlexNet, VGG, WRN [203], LSTM-types [204], GRU [205] on CIFAR-10, Tiny-ImageNet
Targets: Training and inference
Performance Metrics: Quality, compression, instances
Formal Proofs: None

The Lottery Ticket Hypothesis (2018) (LTH) [206] states (and is under certain assumptions proven [207]) that for every neural network there exists a smaller subnetwork which, if trained from scratch, will achieve at least the same accuracy as if the larger super network was trained. Such subnetworks are referred to as winning tickets by LTH. The algorithm provided by the original LTH paper randomly initializing a given DNN, training it for a given number of iterations (e.g., SGD, Adam), prune a certain percentage of parameters by some sensitivity-based heuristic (magnitude is used in the paper) and then re-initialize the remaining weights, creating the winning ticket.
Categories: Sensitivity based unstructured weight pruning
Experiments: LeNet-300-100 on MNIST, scaled down VGG based CNN and VGG-19 on CIFAR-10
Targets: Training (theoretical) and inference (practical) Speedup/Compression
Performance Metrics: Quality, compression
Formal Proofs: None

Online Filter Weakening and Pruning (2018) [208] is a technique to prune the structures of filters as well as filter shapes on CNNs. The authors define constant filter-wise and shape-wise scaling factors to indicate which filters are to be weakened and then train the network from scratch while multiplying the weights with their corresponding scaling factors, which leads to the selected filters and shapes gradually converging to zero, allowing them to be pruned with little loss in accuracy.
Categories: Penalty term based structured pruning of filter-weights
Experiments: VGG-16 on CFIAR-10, WRN-16-4 ans ResNet-164 on CIFAR-100
Targets: Inference
Performance Metrics: Quality, acceleration, compression
Formal Proofs: None

Crossbar-Aware Neural Network Pruning (2018) [113] is a framework specifically tailored to leverage the potential of crossbar architectures (overview see e.g., [209]) through pruning. This is achieved by a newly designed L_0-norm constrained gradient descent (LGD) which takes into account whether the partial sums involved in a convolution's computation efficiently use crossbar memory. The problem of L_0-optimization being an NP problem is mitigated using a relaxant probabilistic projection of the norm.
Categories: Penalty term based structured filter-weight pruning.
Experiments: VGG and ResNet on CIFAR-10 and ImageNet
Targets: Inference
Performance Metrics: Quality, compression, memory
Formal Proofs: None

Qsparse-local-SGD (2018) [210] combines pruning in the form of gradient sparsification, quantization and error compensation with aim of reducing communication overhead in a distributed setting. For quantization, they reduce precision by either randomly or deterministically mapping each of the gradient vectors components to a finite number of

quantization levels. Unstructured sparsification is achieved through the application of quantization.
Categories: Sensitivity based unstructured gradient pruning
Experiments: ResNet50 on ImageNet
Targets: Training
Performance Metrics: Quality, information
Formal Proofs: Convergence

Synaptic Pruning (2018) [211] is a new data-driven approach for CNN pruning based motivated by its biological counterpart [212, 213]. Then, the authors introduce a new class of parameters called synaptic strength in convolutional layers to model how much information a certain connection will contribute to the final result. Synaptic strength is derived as reparametrization of batch normalization, a technique commonly used in convolutional neural networks which, using mini-batch statistics, scales and shifts layer inputs in order to achieve more stable learning. The newly parameterized BN is then optimized by an additional term in the loss function during regular training, and connections with a synaptic strength smaller than a certain threshold are pruned.
Categories: Penalty term and sensitivity based structured filter pruning
Experiments: VGG, ResNet-18, DenseNet-40 on CIFAR-10 and ImageNet
Targets: Inference
Performance Metrics: Quality, compression, operations, instances
Formal Proofs: None

Dropback (2018) introduced by [93] is a DNN training algorithm which trains DNNs under a pruned weight budget. During training, Dropback tracks only the highest k accumulated gradients (k in this context refers to the weight budget) while untracked weights retain their initial values, thus reducing training times and memory accesses significantly. The actual model itself, however, is neither pruned in width nor depth and no advantage for inference is obtained.
Categories: Sensitivity based unstructured weights pruning
Experiments: LeNet-300-100 on MNIST, VGG-S, DenseNet, WRN-28-10 on CIFAR-10
Targets: Inference and training
Performance Metrics: Quality, instances, compression
Formal Proofs: None

Soft filter pruning for accelerating deep convolutional neural networks (2018) (SFP) [214] is an approach for intra-training filter pruning from scratch. SFP prunes filters based on their L_2 norm at the end of each epoch but allows previously pruned filters to be updated in future epochs, thus increasing optimization space and hence accuracy of the such pruned model at the cost of some compression.
Categories: Sensitivity based structured filter pruning
Experiments: ResNet-18/34/50/110 on CIFAR-10 and ImageNet
Targets: Inference

Performance Metrics: Quality, operations, latency, acceleration
Formal Proofs: None

PCONV (2019) [78] proposes to combine coarse-grained structured sparsity with fine-grained unstructured sparsity inside the coarse patterns. Fine-grained sparsity is achieved by applying an enhanced Laplacian of Gaussian filter (approximated through its Taylor expansion), which is also used for image smoothing, edge detection or image sharpening in mathematical vision theory, to convolutional kernels. Coarse grained sparsity is achieved through randomly masking elements of the filter, which can lead to a filter being removed completely.
Categories: Sensitivity based structured and unstructured pruning of filter-weights
Experiments: VGG-16, ResNet-50, MobileNetv2 on CIFAR-10 and ImageNet
Targets: Inference
Performance Metrics: Quality, sparsity, latency, operations
Formal Proofs: None

PruneTrain (2019) [215] focuses on accelerating training of CNN training through structured sparsification via a group lasso regularization and periodical network reconfiguration. Weights are driven towards sparsity through group lasso, grouped at channel granularity and channel where all values are below a certain threshold are pruned. The authors argue that this approach is possible because weights sparsified by group lasso rarely grow above a certain threshold and almost never revive during training.
Categories: Penalty term based structured weights pruning
Experiments: ResNet, VGG on CIFAR-10, CIFAR-100, ResNet on ImageNet
Targets: Training and inference
Performance Metrics: Quality, operations, latency, memory, instances
Formal Proofs: None

A Main/Subsidiary Network Framework for Simplifying Binary Neural Networks (2019) Given that filter level pruning approaches for full-precision NNs cannot simply be ported to binary neural networks (BNNs), a new training approach solving this problem was introduced by [216]. In their framework, a main network learns representative features to optimize prediction performance, while a subsidiary network works as a filter selector on the main network. The subsidiary network, which learns the pruning masks multiplied with the main network's filter-weights, is trained in a layer-wise bottom-up scheme to avoid gradient mismatch.
Categories: Structured filter-weights pruning
Experiments: NIN, VGG-11, ResNet-18 on CIFAR-10, ImageNet
Targets: Inference
Performance Metrics: Quality, operations, acceleration, memory
Formal Proofs: None

Centripetal SGD (2019) (C-SGD) [217] is a novel optimization method seeking to collapse multiple filters into a towards a single point in the hyperspace of the filter parameters (i.e., make filters identical) s.t. then redundant filters can be removed. During training, they divide filters into clusters (evenly or k-means [218]) whereby the number of cluster equals the target number of filters. Then filter updates are computed dependent on an intra-cluster similarity-based constraint: for filters in the same cluster, gradients are averaged and the difference in the initial values is gradually eliminated, leading to filter moving their centre in the hyperspace. Redundant filters are removed after C-SGD training is finished.
Categories: Structured filter-pruning
Experiments: ResNet-56/110/164, Dense-Net-3/6/40, VGG-1/4, VGG-1/2 on CIFAR-10, ResNet-50 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations
Formal Proofs: Convergence

Filter Pruning via Geometric Median (2019) [219] is motivated by an observation similar to what was described by "Rethinking the Smaller-Norm-Less-informative-Assumption" [194] described earlier: whether the norm of a filter is a good heuristic for its informativeness depends on two requirements (large norm deviation, small minimum norm) that are not always met. They propose to solve this problem by pruning filters via geometric median [220] which they use to find filters containing the most replaceable contribution in a layer and prune accordingly.
Categories: Sensitivity based structured filter pruning
Experiments: VGGNet [35], ResNet-20/32/56/110 on CIFAR-10, ResNet-18/34/50/101 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations, compression, latency
Formal Proofs: None

On Implicit Filter Level Sparsity (2019) [221] explores filter-weights sparsity in ReLU-activated CNNs with BN and trained via adaptive gradient descent schemes (Adam [222], Adagrad [223], Adadelta [224]) and L_2-regularizer [2, p. 182] or weights decay. They find that adaptive schemes learn more sparse representations than SGD for a comparable test error, and that the presence of sparsity strictly depends on the presence of regularization and weights decay. The distribution of sparsity across a CNNs layers seems not to be influence by the choice of the optimizer. They further postulate the hypothesis that the higher sparsity observed for training with adaptive optimizers during their experiments stems from the different interaction between adaptive/non-adaptive optimizers and BNs scaling parameters, but no formal proof is provided.
Categories: Penalty term based unstructured filter-weights pruning
Experiments: VGG-16 on CIFAR-10 and CIFAR-100, VGG-11 on ImageNet
Targets: Theoretical insights

Performance Metrics: Quality, compression
Formal Proofs: None

Variational CNN Pruning (2019) [112] is a variational Bayesian scheme for pruning channels in CNNs. The BN layer is reformulated as so-called variational pruning layer by extending the scale factor to the shift term, a concept which the authors refer to as channel saliency. Instead of a deterministic value, however, channel saliency is estimated via variational inference, whereby a sparsity inducing prior is imposed on channel saliency. Redundant channels are then pruned via a threshold criterion based on the distribution of channel saliency.
Categories: Probabilistic sensitivity based structured channel pruning
Experiments: VGG-16, DenseNet-40, ResNet-20/56/110/164 on CIFAR-10, CIFAR-100, ResNet-50 on ImageNet
Targets: Training and inference
Performance Metrics: Quality, instances, operations
Formal Proofs: None

Dynamic Sparse Reparametrization (2019) [225] is a technique for dynamically pruning and growing layers of a CNN given a fixed sparsity parameter as constraint. Every few hundred training iterations, magnitude-based pruning is applied and a certain number of zero-initialized parameters is distributed among sparse tensors, following the rule that layers with a larger fraction of non-zeros receive proportionally more new parameters.
Categories: Sensitivity based unstructured weight pruning
Experiments: WRN-28-2 on CIFAR-10, ResNet-50 on ImageNet
Targets: Training
Performance Metrics: Quality, latency, instances, compression
Formal Proofs: None

Is Non-Structured DNN Weight Pruning Beneficial in Any Platform? (2019) In their paper [77], the authors provide a definitive answer to whether non-structured or structured pruning is most beneficial. They first extend ADMM-NN [226], a joint quantization and (unstructured) weight pruning framework, to perform structured pruning (ADMM-NN-S) and then, through a new fair comparison framework, empirically show that structured pruning always outperforms unstructured pruning for the evaluated architectures and datasets. They further argue pruning should always happen on top of quantization, for the latter being more hardware friendly.
Categories: Penalty term based structured/unstructured weight pruning
Experiments: LeNet-5 on MNIST, AlexNet, ResNet-50 on ImageNet, VGG-16 on CIFAR-10
Targets: Inference
Performance Metrics: Quality, compression
Formal Proofs: None

Eager Pruning (2019) Based on the observation that the ranking of the significance of the weights changes little as training progresses, [227] proposes to start pruning early during training. Their algorithm prunes a certain number of the lowest weights at predefined intervals. Pruning is terminated if loss increases after a pruning step and does not drop back above the average loss of the last n recorded loss values within a specified number of iterations. In such an event, the last pruning step is cancelled. To mitigate inefficient hardware usage which can occur in pruning techniques inducing unstructured sparsity, they further propose a workflow scheduling technique referred to as Dynamically Reconfigurable Add and Collect Tree (DRACT) and design a hardware-efficient DNN architecture around it.
Categories: Sensitivity based unstructured weights pruning
Experiments: ResNet-50, AlexNet, GoogLeNet [228], NIN on ImageNet, ResNet-32 on CIFAR-10, LSTM on TIMIT, LeNet-5 on MNIST
Targets: Training
Performance Metrics: Quality, acceleration, compression, energy, memory
Formal Proofs: None

AutoPrune (2019) Based on the recent work showing that pruning is actually the task of finding the right network structure [229] and thus similar to NAS, AutoPrune [230] prunes the network by optimizing a set of auxiliary variables controlling the network's size instead of the original weights such that the instability and noise during training will have no ill effect on the weight values. They introduce a non-differentiable indicator function which updated separately from the network's weights using straight-through estimation [231] (STE).
Categories: Penalty term based unstructured weights pruning
Experiments: LeNet-5, LeNet-300-100 on MNIST, VGG con CIFAR-10, AlexNet, ResNet, MobileNet [232] on ImageNet
Targets: Training and inference
Performance Metrics: Quality, compression, operations, instances
Formal Proofs: None

Adaptive Group Sparsity based Continual Learning, (2019) (AGS-CL) [233] is a regularization-based continual learning method. Two group-sparsity based penalties are selectively used dependent on each node's importance score. Explicit control of the model's capacity is provided by learning via proximal gradient descent (PGD) [234], which guarantees exact sparsity and model freezing. Further, weights connected to an unimportant node are reinitialized after each update step to ensure efficient learning and prevent the negative transfer in the continual learning scenario.
Categories: Penalty term and sensitivity based structured node and weights pruning.
Experiments: AlexNet on Omniglot [235], CUB200 [236], CIFAR-10, CIFAR-100, MNIST, SVHN [237], Fashion-MNIST [238], Traffic-Signs [239], FaceScrub [240], NotMNIST [241]
Targets: Training

Performance Metrics: Quality, compression, memory, plasticity, stability
Formal Proofs: None

Global Sparse Momentum SGD (2019) [242] is an optimization method aimed at reducing DNNs complexity by on-the-fly pruning. At each training iteration, all parameters are split into two categories and updated using different rules under a global compression constraint. By this updating strategy, redundant parameters are gradually zeroed out by only updating them using ordinary weight decay instead of gradients derived from the objective function.
Categories: Sensitivity based unstructured weights pruning
Experiments: LeNet-300-100, LeNet-5 on MNIST, ResNet-56, DenseNet-40 on CIFAR-10, ResNet-50 on ImageNet
Targets: Training and inference
Performance Metrics: Quality, compression, memory
Formal Proofs: None

Trained Rank Pruning for Efficient Deep Neural Networks (2019) (TRP) [243] recognizes that in approaches separating low rank approximation from training, small approximation errors can lead to disproportionately large network degradation. Hence, they integrate low-rank decomposition in the form of Singular Value Decomposition (SVD) into training by alternating training and SVD, incrementally pushing the weight distribution of a well functioning DNN into a low-rank form while all parameters of the original network are kept and optimized such that its capacity is maintained. They further regularize the network using nuclear norm.
Categories: Penalty term based structured weights pruning
Experiments: ResNet-20/56 on CIFAR-10, ResNet-18 on ImageNet
Targets: Inference
Performance Metrics: Quality, acceleration
Formal Proofs: None

AutoCompress (2020) [244] is an automated pruning framework iteratively employing ADMM to optimize a DNN under certain non-differentiable non-convex structural constraints and pruning the DNN using a purification heuristic. The optimization problem is first split into two separate sub-problems. The DNN is then optimized via gradient descent, while the constraints sub-problem are optimized via Euclidean Mapping. After each round of ADMM, columns and filters are removed if their L_2 norm violates a certain threshold, which is referred to as purification by the authors. Additionally, AutoCompress automatically determines hyperparameters during training via simulated annealing, thus saving the time-consuming trial-and-error process that would otherwise be required to determine the optimal parameter set.
Categories: Penalty term based structured weights, filter and node pruning
Experiments: VGG-16 and ResNet-18/50 on CIFAR-10 and ImageNet,
Targets: Inference

Performance Metrics: Quality, instances, operations, latency
Formal Proofs: None

Harmonious Coexistence of Structured Weight Pruning and Ternarization (2020) [245] is an FPGA/ASIC-focused approach which proposes to orient the target structure of pruning to the underlying hardware's processing elements (PEs) to maximize efficiency. This is achieved by matching the size of group-lassos groups of weights to be pruned with the capacity of the PE. Additionally, quantization in the form of ternerization is integrated into the method. To counteract group-lassos side effect to push weights towards smaller values which would lead to large accuracy degradation, a technique called weight penalty clipping (WPC) is introduced: if the intra-group L_2 norm exceeds a certain threshold, the weights inside the group are considered as too important to be pruned and are spared.
Categories: Sensitivity and penalty term based structured weights pruning
Experiments: ResNet-20/32/44/56 on CIFAR-10, ResNet-18 and AlexNet on ImageNet
Targets: Inference
Performance Metrics: Quality, compression, acceleration
Formal Proofs: None

Structured Sparsification of Gated Recurrent Neural Networks (2020) Advancing structured sparsification of LSTMs and GRUs, [246] proposes to not only sparsify weights and neurons but the pre-activation of the gates as well. This removal of weights by groups corresponding to gates leads to some gates becoming constant and thus allowing simplification of the network's structure. The concept is implemented by applying group-lasso to the weights corresponding to the gates and lasso to individual weights s.t. a multi-level sparsification hierarchy is achieved. Further, during forwards passes, values less than a certain threshold are zeroed out. The network is trained using Bayesian optimization.
Categories: Sensitivity and penalty term based structured weights and neuron pruning
Experiments: Custom LSTMs on IMDB [247], AGNews [248] and PTB [249]
Targets: Inference
Performance Metrics: Quality, compression, instances
Formal Proofs: None

PatDNN (2020) [250] combines non-structured fine-grained pruning with structured coarse grained pruning. In a first stage, which is referred to as pattern-based training by the authors, kernel, and connectivity pruning is applied through an extended ADMM solution framework. In a second stage, the model is converted in into a computational graph and several performance optimization steps are applied: a high-level and fine-grained DNN layer-wise representation, filter kernel reorder, load redundancy eliminations, and automatic parameter tuning.
Categories: Sensitivity and penalty term based structured and unstructured weights and filter pruning
Experiments: VGG-16, ResNet-50, MobileNetV2 on CIFAR-10, ImageNet

Targets: Inference
Performance Metrics: Quality, compression, latency, acceleration
Formal Proofs: None

Non-uniform DNN Structured Subnets Sampling for Dynamic Inference (2020) [251] proposes to train DNNs in such a fashion that each weights tensor contains multiple subnet structures which can be selectively executed at inference time. This is facilitated in two distinct steps: first, subnets are sampled from a complete model using a group lasso with WPC (see above-mentioned [245]) based sampling method and next the subnets identified in the previous step are fused in a single ensemble loss function for multi-objective training.
Categories: Sensitivity and penalty term based structured weights pruning
Experiments: ResNet-20 on CIFAR-10, ResNet-18 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations, latency, acceleration, instances
Formal Proofs: None

FFT-based Gradient Sparsification for the Distributed Training of Deep Neural Networks (2020) Recognizing that existing methods addressing communication overhead in the distributed training of DNNs incur large errors or lose a significant number of gradients near zero, [252] presents a new Fast Fourier Transform (FFT) based gradient sparsification technique which is coupled with range-based variable precision floating-point representation to quantize and compress the gradient frequencies after sparsification. In their framework, gradients are first linearized into 1D vectors for FFT. Next, gradient frequencies are truncated based on their magnitudes to sift out low-energy components and quantized to an N-bit floating-point format. Finally, the now sparse and quantized gradients are transferred to the recipient distributed nodes.
Categories: Sensitivity based unstructured gradient pruning and quantization
Experiments: AlexNet on ImageNet, ResNet-32 on CIFAR-10
Targets: Training
Performance Metrics: Quality, compression, latency
Formal Proofs: Convergence

A Framework for Neural Network Pruning Using Gibbs Distributions (2020) A flexible approach to neural network pruning based on Gibbs distributions is proposed by [253]. Applied with weight magnitudes-based Hamiltonians, they use Gibbs distributions annealing capabilities to smoothly shift from regularization to adaptive pruning. The authors show that their approach is capable of structured and unstructured pruning, dependent on choice of the Hamiltonian.
Categories: Structured/unstructured probabilistic filter and weights pruning.
Experiments: ResNet-20/56 on CIFAR-10
Targets: Inference

Performance Metrics: Quality, compression
Formal Proofs: None

Deep Neural Network Acceleration Based on Low-Rank Approximated Channel Pruning (2020) (LAP) [254] introduces a novel approach to tackle the weight-level redundancy within convolutional filters as well redundant filters themselves. First, a low rank approximation of the DNNs filters is computed by SVD, whereby a spectral norm-based indicator is applied for rank selection. Redundant filters are then removed using Taylor pruning [255], which regards DNN loss as a differentiable function of intermediate activation, and interprets the retained first-order term of the Taylor expansion as an indicator of filter importance. The two steps are applied iteratively during training until a certain compression level is reached, then the network is trained without additional pruning to recover accuracy. The authors further propose Integral of Decay Curve (IDC) based on [256] as a new evaluator for resource constrained scenarios.
Categories: Sensitivity based structured/unstructured filter and filter-weights pruning.
Experiments: AlexNet, ResNet-34/50/152, VGG-16 on ImageNet, CIFAR-10, CIFAR-100, MNIST
Targets: Inference
Performance Metrics: Quality, compression, operations
Formal Proofs: None

Directional Pruning of Deep Neural Networks (2020) [257] brings the Generalization of Regularized Dual Averages [258] (gRDA) algorithm, a gradient-based training approach which has been applied for sparse statistical inference problems with a convex loss function as well as principal component analysis, to DNN training by extending it to mini-batch training. The authors use gRDA to prune weights under the constraint that their projection on to the subspace created by the directions of the second order Taylor expansion of the loss function around a local point in the loss landscape is minimized, which reduces network degradation induced by pruning while training.
Categories: Penalty term based structured weights pruning
Experiments: VGG-16 and WRN-28-10 on CIFAR-100, ResNet-50 on ImageNet
Targets: Inference
Performance Metrics: Quality, compression, memory, latency
Formal Proofs: None

The Generalization-Stability Tradeoff In Neural Network Pruning (2020) [259] illustrate a regularization mechanism which improves DNN generalization exists in pruning independent of its effects on parameter counts and empirically show that magnitude pruning has a similar effect as regularization via noise injection. They further find that this effect on generalization is negatively correlated with the drop in accuracy directly after the application of a pruning step during training, an effect the authors refer to as pruning instability.
Categories: Sensitivity based structured filter and weights pruning

Experiments: VGG-11 and ResNet-18 on CIFAR-10
Targets: Theoretical insights
Performance Metrics: Quality, stability, compression
Formal Proofs: None

Procrustes (2020) [11] is an energy efficient sparse DNN training accelerator aimed at producing pruned models with iso-accuracy. Procrustes is based on above-mentioned Dropback and extends the concept by inducing sparsity through decaying the untracked weights to initial values over the first i iterations s.t. these weights can actually be pruned if they decay to zero, remedying one of Dropback's major drawbacks. Additionally, Procrustes avoids the need to sort all accumulated gradients by using dynamic quantile estimation to continuously track target sparsity.
Categories: Sensitivity/decay based unstructured weights pruning
Experiments: ResNet-18, MobileNetV2 on ImageNet, VGG-S, WRN-28-10, DenseNet on CIFAR-10
Targets: Training
Performance Metrics: Quality, energy, latency, operations, instacnes, compression
Formal Proofs: None

WoodFisher (2020) Based on the classic Optimal Brain Damage/Surgeon (OBD/S) [116, 119] framework, WoodFisher [89] uses second-order information in the form of efficiently approximated (Inverse-) Hessian's to determine the change in loss induced by the removal of one or more parameters and prunes the network architecture accordingly. The approach is notable because not only does it not require retraining after pruning.
Categories: Sensitivity based structured and unstructured weights pruning
Experiments: ResNet-50 on ImageNet, ResNet-20 on CIFAR-10
Targets: Inference
Performance Metrics: Quality, instances, compression
Formal Proofs: None

Network pruning using sparse learning and genetic algorithm (2020) [260] introduces a scheme combining a gradient-limited $L_{1/2}$ regularization during training and an additional channel selection strategy for network pruning. Gradient limited $L_{1/2}$ regularization is applied to the scaling parameter of BN layers s.t. the networks learns to turn off entire channels. The channel selection strategy employs a genetic algorithm which, using the learned channel sparsity from the regularized BN for candidate generation, optimizes each of the layer's pruning mask by a fitness function balancing reduction of FLOPs with loss in accuracy.
Categories: Sensitivity and penalty term based structured channel pruning
Experiments: VGG-16/19, ResNet-20/164 on CIFAR-10/100, VGG-A/16, ResNet-50 on ImageNet, (Slim)YOLOv3 [261, 262] on VOC2007 [263]
Targets: Inference

Performance Metrics: Quality, compression, operations, instances
Formal Proofs: None

Incremental Filter Pruning via Random Walk for Accelerating Deep Convolutional Neural Networks (2020) [264] proposes to prune filters incrementally during training instead of permanently zeroing them, often multiple at once. Instead, in each iteration, L_p norm first is used to evaluate each filter and through a greedy heuristic, select the filter that is to be removed. Then similarly to earlier mentioned SFP, in the next round of training, some previously pruned filters are selected via random walk to be reactivated and allowed to be updated, thus increasing optimization space. This process is repeated until a given sparsity target is met.
Categories: Sensitivity based structured filter pruning
Experiments: ResNet-20/32/56/110 on CIFAR-10, ResNet-18/34/50/101 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations, acceleration, latency
Formal Proofs: None

Winning the Lottery with Continuous Sparsification (2020) [265] employs a new approximation of the otherwise intractable $L0$ regularization to find sparse solutions during DNN training. $L0$ weights regularization is reformulated as $L1$ regularization of the binary pruning mask, which is approximated by the element wise applied Heaviside step function H. To mitigate the problem that H is still intractable, H is approximated by an indexed set of sigmoid functions, which are smooth and differentiable.
Categories: Penalty term based unstructured weights pruning
Experiments: VGG-16 on CIFAR-10, ResNet-20 on ImageNet
Targets: Inference
Performance Metrics: Quality, compression
Formal Proofs: None

Rethinking gradient sparsification as total error minimization (2021) [266] propose a communication complexity model for top-k gradient sparsification (which sends only a subset of gradient coordinates, e.g., [267, 268]) that minimizes the total communication error under a communication budget for the entire training and, motivated by finding that a specific variant of top-k gradient sparsification known as hard-threshold [269, 270] sparsification is optimal in this model, provide convergence analysis for this hard-threshold sparsification in the convex and non-convex case. They further show that hard-threshold has the same asymptotic convergence and linear speed-up property as SGD in both cases and is not affected by data-heterogeneity.
Categories: Sensitivity based unstructured gradient pruning
Experiments: ResNet-18, GoogleNet, SENet-18 [271] on CIFAR-10, LSTM on Wikitext [272], NCF [273] on Movielens-20M [274]
Targets: Theoretical insights

Performance Metrics: Accuracy
Formal Proofs: Convergence

Hybrid Network Compression via Meta-Learning (2021) [275] finds the performance degradation induced by pruning and quantization depends on layer, i.e., different layers can react differently to these two compression techniques. Hence, the authors propose a combination of these two techniques: while the network is pruned by training under $L1$ regularization to induce sparsity, a differentiable integer quantizer is integrated in the network and optimized alongside. Back propagation in the such quantized network is done via STE.
Categories: Penalty term based structured weights pruning
Experiments: VGG-small [276], ResNet-20 on CIFAR-10, MobileNetV2 [40] on Tiny-ImageNet, EDSR [277] on the image super-resolution task
Targets: Inference
Performance Metrics: Quality, information, compression, operations
Formal Proofs: None

Efficient Neural Network Training via Forward and Backward Propagation Sparsification (2021) [278] proposes a completely filter-level sparse training framework for DNNs. The optimization problem of finding filters and corresponding pruning masks is separated into filter update and structure parameter updated and both are optimized through a new non-chain rule-based optimizer referred to as Variance Reduced Policy Gradient Estimator (VR-PGE) which is capable of stochastically estimating gradients during forward passes. VR-PGE avoids dense computations for filters that cannot be reached by the error in the backwards pass due to previous layers being pruned, and thus allows for complete sparse backpropagation.
Categories: Probabilistic structured filter pruning
Experiments: VGG-16/19, ResNet-20 and WideResNet-28-10 [279] on CIFAR-10
Targets: Training
Performance Metrics: Quality, instances, operations, acceleration, latency
Formal Proofs: None

Effective Sparsification of Neural Networks With Global Sparsity Constraint (2021) [280] present a new sparsification method referred to as probabilistic masking (ProbMask), which uses probability as global criterion to measure the importance of all weights in the network. They reformulate training sparse neural networks with L1 norm constraints on each layer's pruning mask as an empirical risk minimization problem, relax it to a continuous and convex expected loss minimization problem and optimize the latter via the PGD method mentioned earlier.
Categories: Probabilistic and penalty term based structured filter and weights pruning
Experiments: VGG-19 and ResNet-32 on CIFAR-10/100, ResNet-50 on ImageNet-1K
Targets: Inference

Performance Metrics: Quality, compression
Formal Proofs: None

Manifold Regularized Dynamic Network Pruning (2021) [281] exploits the manifold information of all samples of a given dataset during the training process to derive instance-specific pruned subnetworks. During training, they first identify each training instance's complexity learning a set of binary variables indicating whether a subnetwork should be thinned out for a specific instance and then adjust each convolutional layer's channel sparsity penalty (expressed as $L1$ norm) correspondingly through a squeeze-excitation control module as introduced in [193]. The subnetworks for each input sample are preserved.
Categories: Penalty based structured channel pruning
Experiments: ResNet-20/32/56 on CIFAR-10, ResNet-18/34, MobileNetV2 on ImageNet
Targets: Inference
Performance Metrics: Quality, operations
Formal Proofs: None

Learning Efficient Convolutional Networks through Network Slimming (2021) [282] propose a training method to induce channel level sparsity. Through multiplying each channel with a scaling factor and jointly optimizing both, with the latter regularized by sparsity inducing $L1$ norm, some channels are driven towards zero during training and are later removed by magnitude. After the removal of weights, a period of fine-tuning follows. This process is applied iteratively until training is terminated by some criteria.
Categories: Penalty and sensitivity based structured channel pruning
Experiments: VGGNet, ResNet, DenseNet on CIFAR-10 and SVHN, VGG-A on ImageNet, custom CNN [150] on MNIST
Targets: Inference
Performance Metrics: Quality, instances, operations
Formal Proofs: None

Enabling Retrain-free Deep Neural Network Pruning Using Surrogate Lagrangian Relaxation (2021) To train a DNN with a loss function subject to constraints on the cardinality of the weights, a problem intractable by regular SGD or analytical methods, [283] reformulates the problem as smaller subproblems which are then solved using Surrogate Lagrangian Relaxation (SLR) [284], a technique similar to earlier mentioned ADMM. After each pruning step, the network is retrained to recover accuracy degradation, which is repeated iteratively until SLR's optimality condition is satisfied.
Categories: Penalty term based structured weights pruning
Experiments: VGG-16, ResNet-18/50/56 on CIFAR-10, ResNet-18/50 and MobileNetV2 on ImageNet, YOLOv3 on COCO
Targets: Inference
Performance Metrics: Quality, compression
Formal Proofs: Convergence

Only train once: A one-shot neural network training and pruning framework (2021) (OTO) [285] By first partitioning the DNNs parameters into zero-invariant groups such that zero-groups can be pruned without affecting the output and then formulating a structured-sparsity optimization problem which is solved via Half-Space Stochastic Projected Gradient (HSPG), a new optimization algorithm, the authors propose a framework for intra-training pruning which does not require any fine-tuning steps or pretrained models.
Categories: Sensitivity and penalty term based structured weights and filter pruning
Experiments: VGG-16 on CIFAR-10, ResNet-50 on ImageNet, Bert [286] on SQuAD [287]
Targets: Inference
Performance Metrics: Quality, instances, operations
Formal Proofs: None

Filter Pruning via Probabilistic Model-based Optimization for Accelerating Deep Convolutional Neural Networks (2021) (FPPMO) [111] observes that most works focusing on filter pruning only consider filter statistics (e.g., norm values) and disregard the behaviour of the pruned network as important signal whether the previous pruning step degraded the network. The authors solve this problem through FPPMO: each filter is assigned a pruning probability, and pruning is conducted during training by sampling masks from the pruning probability distribution. The distribution itself is updated based on the performance of the model in the pruning procedure through minimization of the expected loss under the filter pruning distribution.
Categories: Probabilistic structured filter pruning
Experiments: ResNet-20/32/56/110 on CIFAR-10, ResNet-18/34/50/101 on ImageNet
Targets: Inference
Performance Metrics: Quality, latency, operations
Formal Proofs: None

3.4.1. Analysis of the State-of-the-Art

3.4.1.1. Experiments

To be able to optimally position the new approach introduced in this thesis in the scientific landscape, it is paramount to choose a proper set of experiments such that this work is not only comparable with its direct competitors but also finds acceptance in the wider field of pruning. Hence, the frequency of different architecture-dataset combinations (referred to as experiments in the following) in the literature had to be explored and results are displayed in Table 3.3. The most common experiments by a large margin are training a ResNet architecture on CIFAR-10/100 or ImageNet and a VGG architecture on CIFAR-10/100. Likely, this observation is caused by the simple scalability of these architectures, allowing researches to quickly evaluate different depths of the same architecture, a property also shared by DensNet, MobileNet and WRN architectures. LeNet architectures trained on an MNIST dataset still have a place in modern DNN pruning literature, which is surprising due to this particular combination's age (LeNet was first introduced in 1998). LeNet-MNIST experiments are however simple to set up

and quickly to execute, even on consumer hardware, which might make them attractive to researchers looking for a cheap first evaluation of a new approach. Custom-built NN architectures are used for evaluation only by a small number of works, likely because using non-standard experiments hurts comparability with related work.

	MNIST	**CIFAR**	**ImageNet**
LeNet	11	1	0
AlexNet	2	4	10
ResNet	1	35	37
VGG	1	36	12
WRN	0	8	1
MobileNet	0	2	8
DenseNet	0	8	1
Custom NN	3	1	0

Table 3.3.: Architecture-dataset combinations (experiments) used in recent literature for empirical evaluation. Architecture and dataset subtypes, scales, versions and experiments occurring less than twice are not included due to space constraints. If a scaleable architecture (e.g. ResNet) was used in multiple scales on the same dataset, it was counted once.

The tasks solved by the experiments listed in Table 3.3 are exclusively image recognition tasks. This is notable because the keyword-combinations used during the initial round of literature research did not explicitly exclude other tasks than image recognition. Other tasks from the computer vision domain, such as semantic segmentation, only seem to be of marginal relevance in recent DNN intra-training pruning literature.

3.4.1.2. Metrics

In each experiment conducted in recent literature, a certain set of metrics is used to evaluate the outcome and compare to a baseline or other related work. For this work to be comparable and integrate well into the overall scientific landscape, it is thus necessary to use a subset of metrics most common among other works in the same field. Consequently, the frequency of abstracted metrics was surveyed, and the results are summarized in Table 3.4 on a per-category basis. As illustrated in Table 3.4, using some type of quality metric to determine how well a given task is solved by a pruned DNN is considered essential in the surveyed works and independent of the category they fall into. Besides quality, works targeting inference mostly focus on operations and compression metrics during evaluation. For works targeting training, metrics for evaluating operations and compression are relevant as well; however, instance counts can be observed to play a larger role.

Surprisingly, latency and acceleration seem to be of minor relevance in the field regardless of the category of a given paper.

	QLT	LT	OPS	INS	CMP	ACC	INF	ENG	STB	PLT	MEM
INFER	44	10	24	18	33	13	2	0	0	0	7
TRAIN	13	3	6	7	9	3	1	2	0	0	3
THEO	3	0	0	0	2	0	0	0	1	0	0
PR-ST	5	2	3	2	2	2	0	0	0	0	0
SEN-ST	19	6	13	9	13	8	0	0	1	1	2
PEN-ST	23	5	10	10	19	6	2	0	1	1	5
PR-UST	3	0	0	0	3	0	0	0	0	0	0
SEN-UST	16	4	4	7	14	2	1	1	0	0	2
PEN-UST	7	1	1	3	6	1	0	0	0	0	0
C/F	18	6	15	8	7	4	0	0	1	0	1
W	35	5	14	15	32	9	1	1	1	1	8
G	5	2	1	2	3	0	1	1	0	0	0
N	3	1	1	2	2	0	0	0	0	1	1

Table 3.4.: Frequency of abstracted metrics defined in Table 3.2 per pruning category (descriptions see Section 3.3). Papers can fall into multiple categories and can use multiple metrics. Filter weights pruning is treated as weights pruning. Abbreviations: INFER = Target Inference (e.g., Table 3.7), TRAIN = Target Training (e.g., Table 3.9), THEO = Theoretical insights, PR-ST = Probabilistic Structured, SEN-ST = Sensitivity Structured, PEN-ST = Penalty Term Structured, PR-UST = Probabilistic Unstructured, SEN-UST = Sensitivity Unstructured, PEN-UST = Penalty Unstructured, C = Channel, F = Filter, W = Weights, G = Gradients, QLT = Quality, LT = Latency, OPS = Operations), INS = Instances, CMP = Compression, ACC = Acceleration, INF = Information, E = Energy, MEM = Memory Consumption, PLT = Plasticity, STB = Stability.

3.4.1.3. Hardware Support

While major machine learning frameworks such as PyTorch or TensorFlow in principle support sparse tensor formats and arithmetic, the underlying hardware (i.e., usually GPUs) is often incapable of fully utilizing the potential for acceleration that lies in sparsity. Particularly, unstructured sparsity remains difficult to exploit for all but specialized hardware components [288, 77]. Consequently, significant effort has been devoted to enable hardware to make better use of sparsity, see Section 2.6.5. Some intra-training pruning approaches were developed to adapt the induced sparsity patterns to the underlying hardware's requirements [245, 227, 81] or follow a pattern of algorithm-hardware co-design [11, 79]. Orthogonally, the development of hardware capable of better utilizing sparsity in neural networks is a highly active field of research [289, 290, 291, 292] and automated solutions for the design of such specialized hardware exit as well [293].

Name	Abbreviation	Reference
Variational Pruning	VCP	[112]
SNIP	SNIP	[202]
AutoPrune	AP	[230]
Dynamic Sparse Reparameterization	DSR	[225]
Lottery Ticket Hypothesis	LTH	[206, 207]
Global Sparse Momentum SGD	GSM	[242]

Table 3.5.: Overview of works with possible effect during training but not documented in original paper.

3.4.1.4. Direct Competitors

In this section, I will discuss competitors and their results. While 97.78% of the related work surveyed in section 3.4 introduce improvements for inference (i.e., acceleration, compression), only 28.88% produce at least theoretically some advantageous effect during training (e.g. because pruning is applied very early during training such that the remaining epochs could be accelerated) but do not necessarily document it because the author's main target is to improve inference, e.g., some approaches prune early during training and thus could potentially accelerate the rest of training but no such effect is documented. Only 14.33% actually document such an advantage during training. 6.67% of the surveyed works are of purely theoretically nature. Table 3.5 provides an overview of works providing a potential advantage during training but not documenting it, Table 3.6 provides an overview of works which document a training phase advantage. Since this thesis focuses only documented training phase improvements, works only producing inference phase advantages without even having a theoretically positive impact on training will not be considered competitors. The experimental results of those works which are considered competitors are documented in Tables 3.7, 3.8, 3.9 and 3.10. A visual comparison of the aggregated results of 4 of the 5 competitors listed in Tables 3.9 and 3.10 is provided by Figure 3.3, which indicates PR is the most advantageous approach in terms of acceleration vs. loss of accuracy. However, it must be noted that Figure 3.3 neglects architecture/dataset combinations used in the evaluation of the directly competing works because not all authors conducted the same experiments, so a certain bias induced through the selected experiments must be assumed to be present in Figure 3.3.

Qsparse-local-SGD (QSGD) [210], while showing training phase improvements, is not considered a direct competitor because of its focus on reducing intra node communication overhead in a distributed environment. The training phase acceleration technique FFT-based Gradient Sparsification [252] is as well excluded as a direct competitor due to its focus on a distributed environment. Adaptive Group Sparsity based Continual Learning (AGSCL) [233] is not considered a direct competitor because instead of classification, it solely uses the continual learning task for evaluation.

Name	Abbreviation	Reference
Eager Pruning	EP	[227]
Prunetrain	PT	[215]
Dropback	DP	[93]
Procrustes	PR	[11]
Efficient Neural Network Training	EF	[278]

Table 3.6.: Overview of works specifically targeting training and effect documented in original paper.

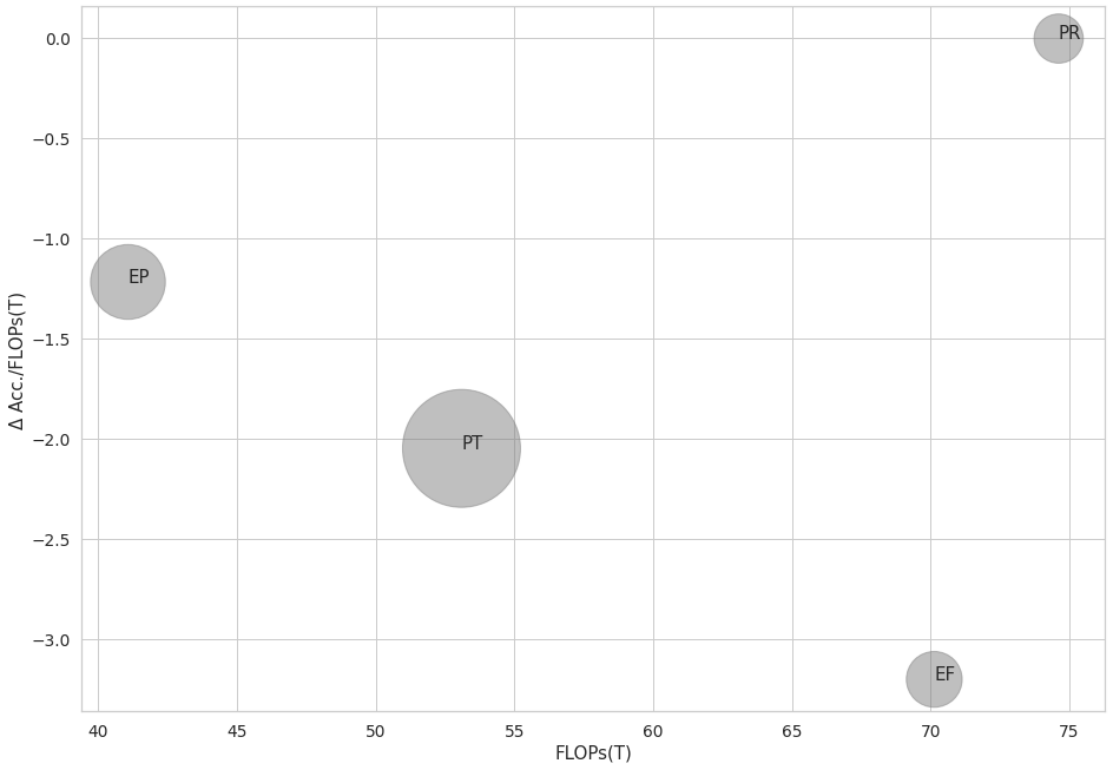


Figure 3.3.: Ball scatter plot across all experiments regardless of architecture/dataset combination listed in Tables3.9 and 3.10. Experiments not reporting training flops decrease neglected (which is why DP is absent in chart). Ball size encodes network density. For PT, which does not report network density, density is assumed to be 100%. Abbreviations: EP = Eager Pruning, PT = Prunetrain, DP = Dropback, PR = Procrustes, EF = Efficient Neural Network Training, see also Table 3.6, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
VCP	CIFAR-10	VGG-16	−0.07 p.p.	62.00%	73.34%	39.10%	?
		DenseNet-40	−0.95 p.p.	60.00%	59.67%	44.78%	?
		ResNet-20	−0.35 p.p.	38.00%	20.41%	16.47%	?
		ResNet-56	−0.78 p.p.	45.00%	20.49%	20.30%	?
		ResNet-110	−0.26 p.p.	63.00%	41.27%	36.44%	?
		ResNet-164	−0.42 p.p.	74.00%	56.70%	49,08%	?
		VGG-16	+0.07 p.p.	32.00%	37.87%	18.05%	?
		DensNet-40	−2.45 p.p.	37.00%	37.73%	22.67%	?
		ResNet-164	−1.80 p.p.	47.00%	17.59%	27.16%	?
	ImageNet	ResNet-50	+0.10 p.p	40.00%	?	?	?
SNIP	CIFAR-10	AlexNet-s	−0.87 p.p.	?	90.00%	?	?
		AlexNet-b	−0.58 p.p.	?	90.00%	?	?
		VGG-C	−0.45 p.p.	?	95.00%	?	?
		VGG-D	−0.33 p.p.	?	95.00%	?	?
		VGG-like	+0.26 p.p	?	97.00%	?	?
		WRN-16-8	−0.42 p.p.	?	95.00%	?	?
		WRN-16-10	−0.52 p.p.	?	95.00%	?	?
		WRN-22-8	+0.29 p.p.	?	95.00%	?	?
	MNIST	LSTM-s	+0.31 p.p.	?	95.00%	?	?
		LSTM-b	−0.20 p.p.	?	95.00%	?	?
		GRU-s	−0.54 p.p	?	95.00%	?	?
		GRU-b	+0.19 p.p	?	95.00%	?	?
		LeNet-300-100	−1.60 p.p.	?	98.00%	?	?
		LeNet-5	−0.80 p.p.	?	99.00%	?	?
AP	MNIST	LeNet-A	−0.22 p.p.[3]	?	?	91.00%	?
			+0.06 p.p.	?	98.75%	?	?
		LeNet-5	−0.02 p.p.[3]	?	?	93.00%	?
			−0.02 p.p	?	99.53%	?	?
			±0.00 p.p	?	99.68%	?	?
	CIFAR-10	VGG-like	−1.90 p.p.[3]	?	?	77.00%	?
			−2.20 p.p	?	98.66%	?	?
	ImageNet	AlexNet	−0.84 p.p	?	94.59%	?	?
		ResNet-50	−0.40 p.p	?	54.54%	?	?

Table 3.7.: Overview of the performance of works with a possible effect during training, but not documented in original paper. Fields with % read as reduced by X%. [3]Neuron Pruning, LeNet-A=LeNet-300-100, VGG-C/D=VGG-16. Abbreviations: VCP = Variational Pruning, SNIP = SNIP, AP = AutoPrune, see Table 3.5, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
LTH	MNIST	LeNet-300-100	?	?	20.00%	?	?
	CIFAR-10	Conv-2	?	10.00%	19.91%	?	?
		Conv-4	?	10.00%	18.92%	?	?
		Conv-6	?	15.00%	16.77%	?	?
		ResNet-18	?	20.00%	19.71%	?	?
		VGG-19	?	20.00%	20.00%	?	?
GSM	MNIST	LeNet-A	±0.00 p.p.	?	98.34%	?	?
		LeNet-5	+0.01 p.p.	?	99.20%	?	?
			−0.15 p.p.	?	99.67%	?	?
	CIFAR-10	ResNet-56	+0.05 p.p.	?	85.00%	?	?
			−0.25 p.p.	?	90.00%	?	?
		DenseNet-40	+0.21 p.p.	?	85.00%	?	?
			+0.16 p.p.	?	87.50%	?	?
			+0.04 p.p.	?	90.00%	?	?
	ImageNet	ResNet-50	−0.39 p.p.	?	75.00%	?	?
			−1.42 p.p.	?	80.00%	?	?
DSR	ImageNet	ResNet-50	−1.60 p.p.	?	90.00%	?	?
			−3.30 p.p.	?	80.00%	?	?
	CIFAR-10	WRN-28-2	?	?	90.00%	?	?
			?	?	80.00%	?	?

Table 3.8.: Overview of the performance of works with a possible effect during training, but not documented in original paper. Fields with % read as reduced by X%. LeNet-A=LeNet-300-100. Abbreviations: GSM = Global Sparse Momentum SGD, LTH = Lottery Ticket Hypothesis, DSR = Dynamic Sparse Reparameterization, see Table 3.5, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
EP	ImageNet	AlexNet	−0.10 p.p.	?	61.69%	?	41.19%
		GoogleNet	−0.80 p.p.	?	70.85%	?	43.25%
		NIN	−1.16 p.p.	?	35.90%	?	26.87%
		ResNet-50	−0.27 p.p.	?	57.81%	?	40.13%
	MNIST	LeNet-5	−0.13 p.p.	?	45.95%	?	38.06%
	CIFAR-10	ResNet-32	+0.13 p.p.	?	66.44%	?	44.86%
	TIMIT	LSTM	−1.71 p.p.	?	72.07%	?	52.40%
PT	CIFAR-10	ResNet-32	−1.80 p.p.	?	?	66.00%	53.00%
		ResNet-50	−1.10 p.p.	?	?	50.00%	70.00%
		VGG-11	−0.70 p.p.	?	?	57.00%	65.00%
		VGG-13	−0.60 p.p.	?	?	56.00%	63.00%
	CIFAR-100	ResNet-32	−1.40 p.p.	?	?	32.00%	46.00%
			−0.70 p.p.	?	?	53.00%	69.00%
		VGG-11	−1.30 p.p.	?	?	47.00%	57.00%
		VGG-13	−1.10 p.p.	?	?	42.00%	52.00%
	ImageNet	ResNet-50	−1.87 p.p.	?	?	40.00%	53.00%
			−1.47 p.p.	?	?	30.00%	44.00%
			−0.24 p.p.	?	?	3.00%	12.00%
DP	MNIST	LeNet-A	−0.10 p.p.	?	81.24%[4]	?	?
			−1.17 p.p.	?	98.12%[4]	?	?
			−2.43 p.p.	?	99.44%[4]	?	?
	CIFAR-10	VGG-S	+0.33 p.p.	?	66.67%[4]	?	?
			+0.18 p.p.	?	80.00%[4]	?	?
			−3.41 p.p.	?	95.0%[4]	?	?
			−10.80 p.p.	?	96.67%[4]	?	?
		DenseNet	+0.62 p.p.	?	77.78%[4]	?	?
			−2.94 p.p.	?	96.30%[4]	?	?
		WRN-28-10	−0.10 p.p.	?	77.78%[4]	?	?
			−0.27 p.p.	?	80.77%[4]	?	?
			−0.45 p.p.	?	86.30%[4]	?	?
	ImageNet	ResNet-18	+1.28 p.p.	?	65.75%[4]	?	?
			+1.64 p.p.	?	82.29%[4]	?	?
			−0.42 p.p.	?	91.14%[4]	?	?
			−7.94 p.p.	?	95.72%[4]	?	?

Table 3.9.: Overview of the performance of works specifically targeting training and effect documented in original paper. Fields with % read as reduced by X%. [4]Not sparsity but tracked parameters during training (i.e., gradient sparsity). LeNet-A=LeNet-300-100, VGG-S=VGG-8, VGG-C/D=VGG-16. Abbreviations: EP = Eager Pruning, PT = Prunetrain, DP = Dropback, see also Table 3.6, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
PR	CIFAR-10	DenseNet	−0.50 p.p.	?	74.37%	70.27%	?[5]
		WRN-28-10	+0.10 p.p.	?	76.94%	78.43%	75.0%
		VGG-S	+0.10 p.p.	?	80.67%	57.99%	?[5]
	ImageNet	MobileNetV2	+0.15 p.p.	?	90.00%	75.08%	74.23%
		ResNet-18	+0.14 p.p.	?	91.45%	80.06%	?[5]
EF	CIFAR-10	VGG-16	−0.30 p.p.	?	96.60%	91.30%	88.49%[6]
		ResNet-20	−0.37 p.p.	?	64.90%	63.90%	52.15%[6]
		WRN-28-10	−0.60 p.p.	?	91.60%	92.10%	89.35%[6]
	ImageNet	ResNet-50	−1.00 p.p.	?	51.80%	53.20%	37.50%[6]
			−3.50 p.p.	?	73.00%	75.30%	66.89%[6]
			−7.70 p.p.	?	89.20%	89.90%	86.41%[6]

Table 3.10.: Overview of the performance of works specifically targeting training and effect documented in original paper. Fields with % read as reduced by X%. LeNet-A=LeNet-300-100. [6]Unit of measurement no specified, FLOPs appears likely though within the context of the paper, [5]Training reduction not provided for all experiments. Abbreviations: EF = Efficient Neural Network Training, PR = Procrustes, see also Table 3.6, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

3.4.2. Scientific Gap and Open Research Questions

Despite pruning in general has seen significant attention during the surveyed period as documented in Table 3.1, many directions remain insufficiently explored in the subdomain of intra-training pruning. As documented in Table 3.5, many works that prune during training only document effects for inference, which presumably could be caused by these methods overhead outweighing any training advantages in terms of runtime or memory complexity and it is unclear how such heuristic overhead can be efficiently mitigated. It has not yet been explored how the interpretability of DNNs changes when they are pruned using the State-of-the-Art methods documented in this chapter, how strong their robustness to fault-injection attacks is or how the combination of these pruning approaches with quantization performs. Furthermore, particularly if the aim is to achieve an acceleration of training, it is unexplored whether accelerating forwards passes through weights pruning and backwards passes through gradient pruning can be improved through the use of separate heuristics for weights and gradients. Whether combining structured and unstructured pruning in intra-training pruning with the aim of accelerating training has not yet been investigated. No method for efficient (and ideally data-agnostic) generation of lottery tickets is currently known and lottery ticket generation consequently cannot practically be employed for training acceleration, only some LTH-inspired but methodically slightly different approaches such as [227] exist. Another topic lacking exploration is how pruning interacts with the vanishing or exploding gradients problem, counter strategies and vice-versa or data-augmentation techniques.

In the following, I will identify concrete open research questions motivated by the documented related work. Each research question will receive a tag in the title of the respective subsection in the form of "Research Question [TAG]" for easier reference later in the thesis.

3.4.2.1. How can computationally expensive pruning heuristics be made more efficient/can computational overhead be reduced? [RQ1]

Works listed in Table 3.7 and 3.8) do not evaluate training despite conceptually, the existence of some training advantages (e.g., speed-up, reduced memory consumption) cannot be excluded. Presumably, this could be caused by the authors either analytically or empirically finding that the advantage in terms of runtime or memory complexity obtained by their approach are offset by their algorithms overhead or, as argued by [278], solely sparsifying weights, while accelerating the forward pass, can still lead to dense backpropagation and hence limit achievable training speed-ups. A possible approach to solve the issue of excessive algorithmic overhead could be of the Monte Carlo type. Hence, one research question I will evaluate in this thesis is whether learning the distribution of the pruning heuristic I will derive for a certain number of epochs and then sampling pruning choices from that distribution is a viable approach to decrease my methods overhead. The viability of such an approach is supported by the observation made by [227, 206] that the importance of weights does not change significantly during training.

3.4.2.2. Does using separate forward- and backwards heuristics yield an advantage over approaches using only one of these two heuristics? [RQ2]

The majority of works listed in Section 3.4 and Table 3.7, 3.8, 3.9 and 3.10 focus on weights pruning, i.e., techniques pruning the networks weights or neurons directly and mostly affecting the forward pass in terms of acceleration and are hence mostly applicable to inference. Few works such as [93] focus on achieving solely a training phase speed up through accelerating the backwards pass (i.e., gradient pruning). Works combining sparse forward and backwards passes do not use separate heuristics (e.g., [93] achieves sparse backwards passes by allowing only small portions of weights to be updated and [11] extends this concept by gradual multiplicative decay of such untracked weights while [278], recognizing the problem of dense backwards passes occurring even if a networks weights are sparse, follows a reverse approach by inducing weights sparsity and zeroing gradients where the weights' sparsity prevents error backpropagation). However, the different characteristics of forwards and backwards pass could necessitate the application of different pruning heuristics: while weights pruning main goal is to leverage sparsity without destroying relevant weights (i.e., already learnt and relevant neural connections shall not be destroyed through pruning), gradient pruning must be conducted in such a fashion that new knowledge is learnt, i.e., only those weight updates should be conducted which will likely have the most positive contribution to the learning effort while the rest is discarded.

3.4.2.3. Does combining structured and unstructured sparsity during training yield an advantage over approaches using only one of these types of sparsity? [RQ3]

Structured pruning generally outperforms unstructured pruning in terms of acceleration and compression due to advantageous exploitation of hardware [77], but unstructured pruning can still be used in combination with structured pruning to increase sparsity further inside the dense substructures that usually exist in structurally-pruned networks [78]. Hence, combined structured and unstructured pruning can be used to obtain an optimal balance between compression, accuracy and acceleration. In works aiming at inference acceleration, this has been successfully demonstrated on multiple occasions (e.g., [78]), but given that none of the works listen in Tabs. 3.9 and 3.10 follows such an approach, it would be novel to bring this concept to training phase acceleration, and hence I will pursue this goal in this thesis.

3.4.2.4. Is there a better yet efficient heuristic for gradient information than norm? [RQ4]

As demonstrated by [194] the norm has undesirable properties limiting its use as a sensitivity heuristic for pruning. Motivated by this, [219] shows geometric median can be used as heuristic for filter pruning, but is aimed at inference. For training, [278], although not relying on a norm-based heuristic, does not use standard chain rule-based SGD for

optimization, [215] uses a penalty term rather than sensitivity heuristics while [93] and [11] both rely on the norm of accumulated gradient for determining the network's sensitivity to certain pruning steps. [227] proposes a pruning algorithm based on weights magnitude instead of norm, but accordingly is only capable of pruning in an unstructured fashion. Hence, the question arises how to prune a network for training acceleration heuristically and in a structured manner using standard SGD for easy integration with popular optimization techniques without relying directly on the norm of the network's weights or gradients.

3.4.2.5. How can lottery tickets be generated efficiently and in a data-agnostic fashion? [RQ5]

The original LTH papers, which are rather of a theoretical nature than of practical applicability, state that a network can be significantly pruned *before* training [206, 207] but they do not examine potential performance gains during training. Motivated by these findings, however, EP [227] seeks a usable realization of LTH's insights by pruning very early during training and demonstrates a speed-up by this technique while [265] employs continuous sparsification during training but only aims at accelerating inference. Still, the question remains unanswered whether an *efficient* lottery ticket generation technique exists, that, before training and possibly without evaluation of training data (i.e., data-agnostic, which EP is not), decides which areas of the network to prune and which not. Efficient in this context is to be understood as that the generation of the lottery tickets shall not incur more overhead than the speed-up that is achieved through training a sparsified network instead of a dense network, i.e., exhaustive training of a large number of differently pruned networks until convergence and then selecting the lottery ticket as done in the original LTH cannot be considered to be an efficient training solution even if the discovered lottery ticket itself produces a training speed-up compared to its dense counterpart.

3.4.2.6. Can a general statement about the the relationship between pruning overhead and obtained training speed-up be made (break-through efficiency)? [RQ6]

As mentioned in Section 3.4.2.1, works listed in Tables 3.7 and 3.8) possibly do not evaluate training phase acceleration due to algorithmic overhead, making improvements during training unlikely. This raises the question of what the relation in intra-training pruning between speed-up (saved OPS) and algorithmic overhead is and whether a generalized formula can be derived making a statement on the minimum reduction of computational complexity of network training any intra-training pruning technique must produce on a given architecture to justify its overhead.

3.4.2.7. Can intra- and post-training pruning methods be combined? [RQ7]

Some works aiming at inference acceleration and compression gather information on weights importance during training and apply a pruning mask after training has finished (e.g., [89] which, besides continuous sparsification during, is also capable of one-shot post-training pruning without retraining). This could make such methods potentially compatible with methods dedicated to achieve intra-training acceleration such as [227], leading to further sparsification and improved inference acceleration.

3.4.2.8. Can existing pruning approaches be combined with quantization? [RQ8]

As suggested by [77], pruning should be used in combination with quantization to achieve most significant improvements in terms of acceleration and compression. Consequently, an interesting subject for further study could be how the combination of training-acceleration focused pruning approaches listen in Table 3.9 and 3.10 fares if combined with intra training quantization approaches such as [23].

3.4.2.9. How does pruning impact the vanishing and exploding gradients problem? [RQ9]

It has not yet been explored how pruned training impacts vanishing/exploding gradients [2, p. 26], counter-strategies [294, 295, 296, 297, 298, 299, 300] and vice versa. It is furthermore unknown how the combination of various approaches performs. It might be worthwhile exploring how a DNN trained under a pruning regime performs if compressed further afterwards through a compression technique (e.g [301]) and whether the results can be improved even more by the most recent data augmentation techniques for semi-supervised and supervised deep learning tasks [302, 303].

3.4.2.10. How robust are pruned DNNs to fault-injection attacks? [RQ10]

Given [304] showed the practical feasibility of software-induced fault injection attacks on DNNs and the high vulnerability of DNNs to such attacks, another important question left untreated by all related work is how these approaches hold up against dedicated attacks as they do not natively include defence mechanisms. Another type of attack to which DNNs are particularly vulnerable are out-of-distribution (OOD) attacks [305] which is why I deem it important to explore how pruning affects a network's robustness to such attacks. Provable guarantees w.r.t. to the detection of OOD attacks are provided by [306].

3.4.2.11. Can abstract abstract guarantees regarding test error be provided for any pruning approach? [RQ11]

Only 10.5% surveyed works provide abstract guarantees regarding convergence, time or space complexity and not a single work provides any guarantees w.r.t. test error (i.e., network degradation induced by pruning). I conjecture component-wise perturbation

analysis [307] might lead to such guarantees, as this technique has in similar scenarios not only been applied to study historical neural network architectures [308, 309, 310] but also yielded results for simple modern architectures recently [311].

3.4.2.12. How is the interpretability of a network affected by pruning? [RQ12]

Another question I deem worthy of further investigation is whether the interpretability [312, 313] of pruned DNNs changes w.r.t. to a base model and whether these changes, if present, can be quantified and lower or upper bounds for their magnitude be provided.

3.4.2.13. Is the conjectured probabilistic/stochastic pruning category of any significance? [RQ13]

As discussed at the beginning of Section 3.3, surveys currently categorize neural network pruning literature along the categories

1. Sensitivity methods
2. Penalty term methods
3. Other methods
4. Information theory methods
5. Structured or unstructured sparsity induction
6. Weight, neuron, filter or layer pruning
7. Type of layer that is pruned

These categories however possibly neglect the existence of a distinctive group of works which rely on probabilistic methods for achieving their goal of pruning neural networks, e.g., [118, 176, 188, 183, 112, 278]. Further research is required to determine whether the introduction of a new category is truly justified and whether it is of any significance compared to other, more dominant categories.

3.4.3. Research Questions Addressed

In this thesis, I will investigate whether a Monte Carlo approach is suitable to reduce the computational overhead of intra-training pruning heuristics (RQ1), I will derive separate heuristics tailored to the specific needs of forwards- and backwards pass (RQ2), I will combine structured and unstructured pruning (RQ3) and further derive a sensitivity heuristics other than norm (RQ4). Additionally, I will explore whether an intra training pruning scheme originally targeted at inference acceleration can be adapted for intra training use (RQ7).

4. Synthesis

The proposed approaches in this thesis are motivated by a subset of the open research questions outlined in Section 3.4.3.

4.1. Pruning Methods

In the following, I will synthesize my pruning approach for accelerating forward and backwards passes. Since multiple combinations of approaches are possible, each approach (and only the concrete approach, not its full mathematical derivation) will be assigned a tag containing and abbreviation for easier later reference. The tags will be given in the title of the respective subsection in the form of "Name of The Approach [TAG]".

4.1.1. Backwards Pass

For accelerating the backwards pass, a heuristic is required to decide which weights are to be updated and where, within a weight's tensor, an update should be applied. Intuitively, this could be achieved via norm, i.e., applying only those weights updates of the gradients with the largest norm, but as discussed in Section 3.4.2.4, norm is generally of limited usefulness for pruning and for the special case of pruning gradients during the backwards pass, just updating those weights where the norm of the gradients at a given batch are the largest fails to capture the trajectory of gradients over multiple batches.

4.1.1.1. Global Gradient Diversity GGD

Gradient Diversity (GD) was introduced by [25] as a global (hence GD is also referred to as GGD here) measure to quantify the degree to which individual gradients at a given batch are different w.r.t to previous batches and was successfully applied by [64] as heuristic to decide when a precision switch should happen during intra-training quantization. For look back lb, which denotes the number of previous batches that are used for computation, GGD is given by

$$\Delta s(\boldsymbol{W})_{lb} = \sum_{l \in L} \frac{\sum_{k=0}^{lb} \|\nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)\|_2^2}{\|\sum_{k=0}^{lb} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)\|_2^2} \tag{4.1}$$

GGD is large when the inner products between the gradients taken with respect to different batches are small, which, as shown by the authors in an extended version [25, arXiv: 1706.05699v3] of their paper on gradient diversity, is the case when the gradients are almost orthogonal, or even on opposite directions. The case that this is a desirable

property for pruning is supported by previous works relying on heuristics incorporating the directions of gradients [89, 116, 119] as well as novel insights into the relation of the gradients directions with the geometry of the decision surface manifold [314].

4.1.1.2. Layerwise Gradient Diversity LGD

The concept of GGD was later refined to LGD by [23] and used in a local, layer-wise applied heuristic for precision switches in intra-training quantization on a per-layer basis. For a layer l, look back lb, LGD is given by

$$\Delta s(\boldsymbol{W}^l)_{lb} = \frac{\sum_{k=0}^{lb} \left\| \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2^2}{\left\| \sum_{k=0}^{lb} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2^2} \tag{4.2}$$

which is connected to GGD by

$$\Delta s(\boldsymbol{W})_{lb} = \sum_{l \in L} \Delta s(\boldsymbol{W}^l)_{lb}$$

4.1.1.3. Efficient Computation In the Normalized Gradients Case

This section introduces a novel computation technique for LGD and GGD and analytically argues for its improved memory efficiency. Previously used computations of gradient diversity required the storage of the gradients of the last lb batches of L layers, causing excessive memory consumption for large values of lb. A common technique for combating the vanishing and exploding gradients problem [2, p. 129-134] though is gradient normalization [2, p. 142] [315, 316]. For normalized gradients,

$$\widehat{\nabla}^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) = \frac{\nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)}{\left\| \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2} \tag{4.3}$$

both LGD and GGDs memory complexity can be greatly simplified. LGD as defined in Equation (4.2) becomes

$$\Delta s(\boldsymbol{W}^l)_{lb} = \frac{lb}{\left\| \sum_{k=0}^{lb} \widehat{\nabla}^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2^2} \tag{4.4}$$

because

$$\left\| \widehat{\nabla}^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2 = 1 \forall l, k \Rightarrow \sum_{k=0}^{lb} \left\| \widehat{\nabla}^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2^2 = lb$$

and GGD as defined in Equation (4.1) becomes,

$$\Delta s(\boldsymbol{W})_{lb} = \frac{L \cdot lb}{\sum_{l \in L} \left\| \sum_{k=0}^{lb} \widehat{\nabla}^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2^2}$$

whereby the accumulated gradients used in $\left\|\sum_{k=0}^{lb} \widehat{\nabla}^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)\right\|_2^2$ can be computed on the fly at the end of each batch such that storing lb gradients becomes unnecessary, greatly reducing storage requirements of GD. This means if one leverages gradient normalization as a tool for combating vanishing and exploding gradients, it is possible to get LGD and GGD for free (almost) as no additional memory except for storing the accumulated gradients has to be allocated. $\mathcal{O}(lb \cdot m)$ becomes $\mathcal{O}(m)$ where m is the memory required to store the network's accumulated gradients, which is equal to the storage requirements of the network's trainable weights themselves. This is a serious improvement, as shall be demonstrated empirically in Section 6.3.3.1, rendering previously intractable problems tractable.

4.1.1.4. Intra-Layerwise Gradient Diversity (ILGD)

Technically, it is possible to take GD to an even more fine granular level than just layerwise computation by computing its value for specific areas of a layer's gradient tensor. Since weights in a convolutional layer are organized in a 4D tensor and weights in a linear layer are organized in a 2D matrix [317], we obtain for linear layer l, look back lb, coordinates h, w

$$\Delta s(\boldsymbol{W}^l)_{lb,(h_i,w_i)} = \frac{\sum_{k=0}^{lb} \left\|\nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(h_i,w_i)}\right\|_2^2}{\left\|\sum_{k=0}^{lb} \nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(h_i,w_i)}\right\|_2^2} \tag{4.5}$$

or convolutional layer filters with in- and out channels n, c

$$\Delta s(\boldsymbol{W}^l)_{lb,(n_i,c_i,h,w)} = \frac{\sum_{k=0}^{lb} \left\|\nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(n_i,c_i,h,w)}\right\|_2^2}{\left\|\sum_{k=0}^{lb} \nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(n_i,c_i,h,w)}\right\|_2^2} \tag{4.6}$$

Potentially ILGD could be a suitable heuristic for weights pruning. Unfortunately though, the efficient computation introduced in the Section 4.1.1.3 for LGD and GGD is not applicable for ILGD because even if normalized gradients with $\left\|\widehat{\nabla}^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)\right\|_2 = 1$ are used it holds that

$$\exists h_i, w_i | \sum_{k=0}^{lb} \left\|\nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(h_i,w_i)}\right\|_2^2 \neq L_l$$

and

$$\exists n_i, c_i, h, w | \sum_{k=0}^{lb} \left\|\nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)_{(n_i,c_i,h,w)}\right\|_2^2 \neq L_c$$

with L_l, L_c denoting the number of linear and convolutional layers respectively which again would necessitate extensive storage of lb gradients.

4.1.1.5. Top-K Gradients with LGD [GDTopK]

Given that ILGD as provided in Equations (4.5) and (4.6), while desirable in terms of granularity, is still intractable in terms of memory complexity, I resort to an efficient LGD (Equation (4.4)) based top-k gradients scheme for accelerating the backwards pass similar to what is used in [93, 11] where only the most important gradients are actually used in the backwards pass to update the networks weights, saving operations in the process. With LGD I use a different and potentially advantageous metric though and prune entire gradients layerwise instead of areas of gradients due to the intractability of ILGD for large networks. The proposed gradient pruning scheme is illustrated in Algorithm 3. On initialization, Algorithm 3 receives the arguments *lb*, network, loss, epochs, and the number of gradients to be deleted k (lines 1-2). Next for each batch in each epoch, prediction, loss and gradients are computed (lines 3-5) normally as described in Section 2.4, gradients are normalized (line 6) as shown in Equation (4.3) and accumulated (line 7) for batch k i.e.

$$\forall l \nabla^{(l,acc)}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) += \nabla^{l}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k). \tag{4.7}$$

Then, if *lb* gradients have been accumulated, LGD is computed as described in Equation 4.4 and the accumulated gradient $\forall l \nabla^{(l,acc)}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)$ is reset to all zeros (lines 8-10). The algorithm next proceeds top delete the k gradients with the lowest LGD value (line 12) and the weights are updated by the optimizer as described in Section 2.4.2 using the pruned gradients (line 13).

```
   Data: lb, network, loss epochs, data, k
 1 for epoch in epochs do
 2 |   for batch, labels, index in data do
 3 |   |   prediction = network.forward(batch)
 4 |   |   loss = loss(prediction, labels)
 5 |   |   gradients = computeGradients(loss)
 6 |   |   normalizedGradients = normalizeGradients(gradients)
 7 |   |   accumulateGradients(normalizedGradients)
 8 |   |   if index mod lb = 0 then
 9 |   |   |   lgd = updateLGD(accumulatedGradients)
10 |   |   |   resetAccumulatedGradients(accumulatedGradients)
11 |   |   end
12 |   |   prunedGradients = deleteGradients(normalizedGradients, lgd, k)
13 |   |   network.backwards(prunedGradients)
14 |   end
15 end
   Result: network
```

Algorithm 3: Top-K Gradients with LGD Pseudo Code [14]

4.1.1.6. Monte-Carlo Top-K Gradients with LGD [GDTopKMC]

While LGD as provided by Equation (4.4)is already efficient, its computation still has non-zero overhead. To save the overhead of computing the LGD heuristic every lb batches, one possible approach is to learn the probabilities of individual layers getting pruned by the heuristic over a given number of sample epochs se and then draw from this distribution during further training epochs. Best suited for such a task is the categorical distribution. The categorical distribution, which generalizes the Bernoulli distribution to a categorical random variable, is a discrete probability distribution with a sample space of a given set of categories. Let $\boldsymbol{X}$ be a categorical distribution for a set of C categories, $|C| = n$, then $\boldsymbol{X}$ is completely characterized by the vector of probabilities $\boldsymbol{p}$ of the associated categories $c_i \in C$ with $\boldsymbol{p} = p_1, \ldots, p_n$ and $\sum_{i=1}^{n} p_i = 1$. Updating $\boldsymbol{X}$ based on new observations can now be accomplished by updating individual probabilities p_i

$$p_i = p_i + \frac{(obs_i - p_i)}{N} \tag{4.8}$$

where observed probabilities obs_i are described by

$$obs_i = \frac{\sum_{j=1}^{k} [o_j = c_i]}{k} \tag{4.9}$$

with N being the total numbers of observations, k denoting the number of observations since the last update of $\boldsymbol{X}$ and $[o_j = c_i]$ denoting the event that c_i was observed.

To apply Equations (4.8) and (4.9) to the case of gradient pruning, let L be a set of layers of a neural network, $P(L)$ an associated pruning heuristic which, for each call, selects a single element $l \in L$ for gradients pruning, e.g., top-k gradients with LGD as described in Section 4.1.1 Equation (4.4) and Algorithm 3. Further, let lb be the lookback as described in Section 4.1.1.2 and let e be the current epoch, $emax$ the total training epochs, bs be batch size, bd be the number of batches in the dataset, se be the number of sampling epochs. Then $\boldsymbol{X}_L$ with characterizing probabilities vector $\boldsymbol{p}_L$ describes the probability distribution of a layer $l \in L$ getting selected by $P(L)$. At the beginning of training, $\boldsymbol{X}_L$ can be initialized with $p_i = \frac{1}{|L|} \forall p_i \in \boldsymbol{p}_L$, i.e., each layer starts with the same probability of being selected by the pruning heuristic because no observations are yet available. As training progresses, the elements of $\boldsymbol{p}_L$ is updated every sampling epoch se by

$$p_i = p_i + \frac{(obs_i - p_i)}{N} \tag{4.10}$$

$$N = \frac{bd \cdot e}{bs \cdot lb \cdot se}$$

$$obs_i = \frac{\sum_{j=1}^{db/lb} [P(L)_j = l]}{bd/lb} \tag{4.11}$$

During sampling epochs, the actual heuristic is computed and layers pruned accordingly. Otherwise, the decision which layers to prune are drawn from $\boldsymbol{X}_L$ whereby to pruned k gradients, it is necessary to draw k times from $\boldsymbol{X}_L$.

The integration into training is given by Algorithm 4. On initialization, Algorithm 4 receives the arguments *lb*, network, loss, epochs, the number of gradients to be deleted k as well as the number of epochs used for sampling *se*. Next for each batch in each epoch, prediction, loss and gradients are computed (lines 3-5) normally as described in Section 2.4, gradients are normalized (line 6) as shown in Equation (4.3). Then, if the current epoch is a sampling epoch, gradients are accumulated (line 7-8) for the batch k according to Equation (4.7). If *lb* gradients have been accumulated, LGD is computed as described in Equation 4.4, the probabilities of the categorical probability distribution are updated as shown in Equations (4.10) and (4.11) and the accumulated gradient $\forall l \nabla_{\boldsymbol{W}_k}^{(l,acc)} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)$ is reset to all zeros (lines 9-12). The algorithm in line 14 proceeds to delete k gradients deterministically using identical mechanics as Algorithm 3. Otherwise, if the current epoch is not a sampling epoch, the k gradients that are deleted are determined by drawing k times from the categorical probability distribution (line 16). Finally, the weights are updated by the optimizer as described in Section 2.4.2 using the pruned gradients (line 18).

```
   Data: lb, network, loss epochs, data, k, se
 1 for epoch in epochs do
 2     for batch, labels, index in data do
 3         prediction = network.forward(batch)
 4         loss = loss(prediction, labels)
 5         gradients = computeGradients(loss)
 6         normalizedGradients = normalizeGradients(gradients)
 7         if epoch mod se = 0 then
 8             accumulateGradients(normalizedGradients)
 9             if index mod lb = 0 then
10                 lgd = updateLGD(accumulatedGradients)
11                 probs = updateProbabilities(lgd, k)
12                 resetAccumulatedGradients(accumulatedGradients)
13             end
14             prunedGradients = deleteGradients(normalizedGradients, lgd, k)
15         else
16             prunedGradients = deleteGradientsMC(probs, k)
17         end
18         network.backwards(prunedGradients)
19     end
20 end
   Result: network
```

Algorithm 4: Monte Carlo Top-K Gradients with LGD Pseudo Code

4.1.2. Forwards Pass

4.1.2.1. Kullbeck-Leibler Divergence (KLD)

Determining the amount of information loss, if a layer's weights tensor is perturbed, can be heuristically accomplished by interpreting the perturbation as a change of encoding. Assume a weight's tensor $\boldsymbol{W}^l \sim Q^l$ of layer $l \in \mathbb{L}$ with its perturbed counterpart $\widehat{\boldsymbol{W}}^l \sim P^l$, where P^l, Q^l are the respective distributions. Then the continuous Kullback-Leibler-Divergence introduced by [26] represents the average number of bits lost through changing the encoding of l from Q^l to P^l, with p and q denoting probabilities, and w the elements of the weight's tensor:

$$D(P^l \| Q^l) = \int_{-\infty}^{\infty} p^l(w) \cdot log \frac{p^l(w)}{q^l(w)} dw$$

Using discretization via binning, we obtain P^l and Q^l at resolution r^l through the empirical distribution function:

$$\widehat{F_{r^l}}(w) = \frac{1}{r^l + 1} \sum_{i=1}^{r^l} 1_{W_i^l \leq w} \tag{4.12}$$

that can then be used in the discrete Kullback-Leibler-Divergence.

$$KL(P^l \| Q^l) = \sum_{w \in \boldsymbol{W}^l} P^l(w) \cdot log \frac{P^l(w)}{Q^l(w)} \tag{4.13}$$

4.1.2.2. Measuring Learned Information with KLD

Assume standard non-nesterov SGD without momentum, dampening, and constant learning rate $\eta = \boldsymbol{const}$ (see Section 2.4.2.1). Then as described in Equation (2.9) an update step at batch j is given by:

$$\boldsymbol{W}^l_{(j+1)} = \boldsymbol{W}^l_j - \eta \cdot \nabla^l_{\boldsymbol{W}_j} \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j)$$

Which allows me in the accumulated lb gradients' case to obtain the update step via

$$\boldsymbol{W}^l_{(j+1)} = \boldsymbol{W}^l_{(j-lb)} - \sum_{k=j-lb}^{j} \eta \cdot \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) = \boldsymbol{W}^l_{(j-lb)} - \eta \cdot \sum_{k=j-lb}^{j} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \tag{4.14}$$

whereby the right-hand side of the term is efficiently computable without storing individual gradients of each batch, just the accumulated gradients need to be stored. Using Equations (4.12), (4.13) and (4.14), the information change Γ in $\boldsymbol{W}^l$ induced by the last lb gradients in layer l is then given by:

$$\Gamma^l = KL(F_{r^l}(\boldsymbol{W}^l_{(j+1)}) || F_{r^l}(\boldsymbol{W}^l_{(j-lb)})) =$$

$$KL(F_{r^l}(\boldsymbol{W}^l_{(j+1)})) || F_{r^l}(\boldsymbol{W}^l_{(j+1)} + \eta \cdot \sum_{k=j-lb}^{j} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)) \tag{4.15}$$

or with respective formulations $\Gamma^l_{n_i,c_i,h,w}$ for intra convolution and $\Gamma^l_{h_i,w_i}$ intra linear layer application (analogue to ILGD's Equations (4.5) and (4.6) in Section 4.1.1.4). Observe the right-hand side can be computed just from the accumulated gradients that is already computed for GD as described in Section 4.1.1 as well, so no need to additionally store old weights arises if Γ^l is used in conjunction with a GD-based heuristic.

4.1.2.3. Approximation via Relative Error

In practice, I found though that because sufficient binning size is required, the KL-based heuristic in Equation 4.15 is not applicable to pruning small filters in convolutional layers (e.g., 5x5 filter = max 25 bins, usually much less sensibly, not enough). I propose the following simplification through approximation of above metric via relative error:

$$\widehat{\Gamma}^l = \left\| 1 - \frac{\left\| \boldsymbol{W}^l_{(j+1)} + \eta \cdot \sum_{k=j-lb}^{j} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) \right\|_2}{\left\| \boldsymbol{W}^l_{(j+1)} \right\|_2} \right\|_2 \tag{4.16}$$

The heuristic $\widehat{\Gamma}^l$ contains similar information as Γ^l but is cheaper to compute and performs better in practice due to not being dependent on binning in cases of respective formulations $\widehat{\Gamma}^l_{n_i,c_i,h,w}$ for intra convolution and $\widehat{\Gamma}^l_{h_i,w_i}$ intra linear layer application.

In order to find areas of a layers weights tensor to prune via Equation (4.16)'s intra-layer formulations, the proper boundaries n_i, c_i, h, w and h_i, w_i have to be found. This is achieved by introducing two scaling parameters σ_c and σ_l and sampling ζ_c and ζ_l times respectively from respective discrete uniform distributions, i.e., $n_i \in [1, \sigma_c \cdot n]$ and $c_i \in [1, \sigma_c \cdot c]$ in the convolutional case and $h_i \in [1, \sigma_l \cdot h]$ and $w_i \in [1, \sigma_l \cdot w]$ the linear case. After sampling, the areas of the convolutional and linear tensors associated below κ_c, κ_l quantiles are zeroed out via application of a binary pruning mask $\mathbf{M}$.

4.1.2.4. Relative Error Pruning [REP]

The relative error approximated KL pruning heuristic introduced in Section 4.1.2.3, which I abbreviate as *RE-Pruning*, is implemented in the eager fashion motivated by [227, 206, 207], i.e., the networks weights and gradients are only aggressively pruned early during training. Integration into training of is achieved as shown in Algorithm 5. On initialization, Algorithm 5 receives the arguments *lb*, network, loss, epochs, pruning epochs *pe*, pruning parameters σ_c, σ_l, κ_c, κ_l, ζ_c, ζ_l and threshold parameters τ_c, τ_l (lines 1-2). Next for each batch in each epoch, prediction, loss and gradients are computed (lines 3-5) normally as described in Section 2.4 and gradients are normalized (line 6) as shown in Equation (4.3). If then $epoch \leq pe$, gradient accumulation for batch k is triggered as shown in Equation (4.7) (lines 7-8). Once *lb* gradients have been accumulated, the pruning mask according to the heuristic described in Section 4.1.2.3 is computed, the accumulated gradient $\forall l \nabla^{(l,acc)}_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)$ is reset to all zeros and the pruning mask is applied to the networks weights and gradients (lines 10-13). The case $epoch > pe$ is handled in lines 14-15: after the aggressive eager pruning phase for the first *pe* epochs of training where structured

sparsity is induced is over, the network is only further pruned by magnitude pruning. The intuition behind this choice is twofold: for one, magnitude pruning additionally applies unstructured pruning over the structures induced during the eager pruning phase, further enhancing sparsity. Additionally, because magnitude pruning does not have any structural constraints, it allows some particularly important weight elements which were unintentionally pruned during the structured pruning phase to grow back, hence allowing the repair of the damage induced through structural pruning. For this strategy to work optimally, however, the loss function must be regularized such that a convergence of the network towards large weights is inhibited. For the experiments conducted in Section 6, I chose a linear combination of L_1 and L_2 regularization as described in Section 2.4.3. Finally, the now pruned weights are updated by the optimizer as described in Section 2.4.2 using the pruned gradients (line 17).

```
   Data: lb, network, loss, epochs, data, pe, pruningParameters, thresholdParameters
 1 for epoch in epochs do
 2     for batch, labels, index in data do
 3         prediction = network.forward(batch)
 4         loss = loss(prediction, labels)
 5         gradients = computeGradients(loss)
 6         normalizedGradients = normalizeGradients(gradients)
 7         if epoch ≤ pe then
 8             accumulateGradients(normalizedGradients)
 9             if index mod lb = 0 then
10                 mask = computeMask(accumulatedGradients, pruningParameters)
11                 resetAccumulatedGradients(network.weights, accumulatedGradients)
12             end
13             network.weights, prunedGradients = applyMask(mask, network.weights,
                normalizedGradients)
14         else
15             network.weights, prunedGradients = applyThreshold(network.weights,
                normalizedGradients, thresholdParameters)
16         end
17         network.backwards(prunedGradients)
18     end
19 end
   Result: network
```

Algorithm 5: RE-Pruning Pseudo Code

4.2. Combined Approaches

Not only can the above presented methods REP for forward and GDTopK, GDTopKMC backwards pass pruning be freely combined with one another, it is further possible to

combine them with certain other DNN training acceleration techniques seamlessly.

4.2.1. Alternating Direction of Multipliers (ADMM) [ADMMR, ADMMI]

A systematic framework for DNN weight pruning based on ADMM and aimed at accelerating inference was proposed by [18] (also see Section 3.4). The optimization problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + g(\boldsymbol{x})$$

with f differentiable and g non-differentiable can be rewritten to

$$\min_{\boldsymbol{x},\boldsymbol{z}} f(\boldsymbol{x}) + g(\boldsymbol{z}) \text{ subject to } \boldsymbol{x} = \boldsymbol{z}$$

which through augmented Lagrangian [318] can be split into two subproblems $\boldsymbol{x}$ and $\boldsymbol{z}$. The first problem is $\min_{\boldsymbol{x}} f(\boldsymbol{x}) + q_1(\boldsymbol{x})$ and the second problem $\min_{\boldsymbol{z}} g(\boldsymbol{z}) + q_2(\boldsymbol{z})$ whereby $q1$ and $q2$ are quadratic functions of their arguments and the first problem is differentiable. In special cases, problem two might be solved analytically by exploiting special properties of g, e.g., if g is a regularizer such as L_2, L_1 or non-differentiable $L0$ which allows formulating these sparsity requirements as combinatorial constraints with a unique solution. ADMM repeats these steps and optimizes both subproblems until a predefined number of iterations is reached. The integration of ADMM into the pruning framework proposed by [18] is implemented by splitting training in three phases as illustrated by Algorithm 6: first, a pre-training phase is conducted where the network is optimized regularly (e.g., SGD, Adam, see Section 2.4.2) for a certain number of epochs without any additional constraints. Next, a pruning-phase follows during which the above described ADMM algorithm is executed for a given number of epochs s.t. the network is sparsified. Finally, the now sparse network is retrained using regular gradient-based optimization again but under the constraint that previously zeroed-out elements are fixed, i.e., the network is prevented from converging towards a dense solution.

The integration of the penalty term type ADMM pruning framework with my proposed pruning methods can be done two-fold: either keep the schedule of pre-train (Algorithm 6 lines 1-7), prune (Algorithm 6 lines 9-19), re-train (Algorithm 6 lines 20-28) as proposed by the original authors but add my proposed top-k gradients method for backwards pass acceleration (Section 4.1.1.5) and my proposed weights pruning method for forwards pass acceleration (Section 4.1.2.4) to the pre- and -retraining and prune phases or reschedule pre-train, prune, re-train phases originally proposed such that they are shortened but repeated many times over (Algorithm 7) in combination with my proposed approaches. The advantage of rescheduling the three phases of pruning lies in the earlier generation of sparsity, which allows for a lengthier period of training where it can be exploited for acceleration. The motivation for repeating short pre-train, prune, re-train cycles many times over instead of simply shortening pre-train and pruning cycles in favour of long retraining phases comes from the observation that reducing the total amount of pruning epochs reduces the degree of sparsity.

```
Data: network, loss, epochs, data,
pre epochs, admm epochs, re epochs
for epoch in pre epochs do
    for batch, labels, index in data do
        prediction = network.forward(batch)
        loss = loss(prediction, labels)
        gradients = computeGradients(loss)
        network.backwards(normalizedGradients)
    end
end
for epoch in admm epochs do
    for batch, labels, index in data do
        prediction = network.forward(batch)
        loss = admmLoss(prediction, labels)
        gradients = computeGradients(loss)
        network.backwards(gradients)
    end
    updateDualVariables()
end
mask = computeMask(network)
applyMask(mask, network.weights)
for epoch in re epochs do
    for batch, labels, index in data do
        prediction = network.forward(batch)
        loss = loss(prediction, labels)
        gradients = computeGradients(loss)
        prunedGradients = applyGradientMask(gradients)
        network.backwards(prunedGradients)
    end
end
Result: network
```

Algorithm 6: ADMM Original Training Scheme [18] Pseudo Code [ADMMR]

4.2.2. Gradient Accumulation [AC]

A common practice to accelerate gradient-based deep learning applications which is often employed in conjunction with quantization, pruning or distributed training is gradient accumulation (e.g., [23, 268, 319, 320]). In principle, the idea of gradient accumulation can, in the simplest case of SGD, be mathematically described as augmenting the weight update Equation (2.9)

$$\boldsymbol{W}^l_{(j+1)} = \boldsymbol{W}^l_j - \eta \cdot \nabla^l_{\boldsymbol{W}_j} \mathcal{L}(\boldsymbol{W}_j; \boldsymbol{y}_j, \boldsymbol{X}_j)$$

```
Data: network, loss, epochs, data,
pre epochs, admm epochs, re epochs, repetitions
for repetition in repetitions do
    for epoch in (pre epochs)/repetitions do
        for batch, labels, index in data do
            prediction = network.forward(batch)
            loss = loss(prediction, labels)
            gradients = computeGradients(loss)
            network.backwards(normalizedGradients)
        end
    end
    for epoch in (admm epochs)/repetitions do
        for batch, labels, index in data do
            prediction = network.forward(batch)
            loss = admmLoss(prediction, labels)
            gradients = computeGradients(loss)
            network.backwards(gradients)
        end
        updateDualVariables()
    end
    computeMask(network)
    applyWeightMask(network)
    for epoch in (re epochs)/repetitions do
        for batch, labels, index in data do
            prediction = network.forward(batch)
            loss = loss(prediction, labels)
            gradients = computeGradients(loss)
            prunedGradients = applyGradientMask(gradients)
            network.backwards(prunedGradients)
        end
    end
end
Result: network
```

Algorithm 7: ADMM Proposed Training Scheme Pseudo Code [ADMMI]

with the accumulation of the last ac gradients (similar as in Equation (4.14))

$$\boldsymbol{W}^l_{(j+1)} = \boldsymbol{W}^l_{(j-ac)} - \sum_{k=j-ac}^{j} \eta \cdot \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k) = \boldsymbol{W}^l_{(j-ac)} - \eta \cdot \sum_{k=j-ac}^{j} \nabla^l_{\boldsymbol{W}_k} \mathcal{L}(\boldsymbol{W}_k; \boldsymbol{y}_k, \boldsymbol{X}_k)$$

whereby an update is only executed every ac batches. Thus the operations otherwise required for $ac - 1$ weight update steps are saved in the process. Due to its simplicity, gradient accumulation can be integrated seamlessly into training with both my proposed

pruning heuristics.

4.2.3. Magnitude Based Unstructured Pruning [MAG]

Previously mentioned pruning methods solely induce different types of structured sparsity. This is all good for performance, but for maximum compression, unstructured sparsity is required as well. This can be reached through simply adding magnitude pruning, whereby weights below a certain threshold are cut off. While this might a naive approach, it will be evaluated as part of REP (Section 4.1.2.4).

4.2.4. Using the Feedback Provided by Loss Function [DK]

The importance of incorporating the feedback provided by the loss function into pruning was recognized by [111]. Motivated by these findings, I decided to incorporate the loss functions feedback into by making k and ac in Top-K gradients (see Sections 4.1.1.5, 4.1.1.6) with gradient accumulation (Section 4.2.2) dependent on the loss of the rolling average of the previous lb batches. For ac at batch i and growth factor ω, the new accumulation value is obtained via:

$$ac_i = \begin{cases} max(1, ac_{i-1} \cdot (1-\omega)) & \text{if } \frac{\sum_{j=0}^{lb} loss_j}{lb} > loss_i \\ min(lb - i \bmod lb, ac_{i-1} \cdot (1+\omega)) & \text{else if } k < k_{max} \end{cases}$$

For k, first a similar mechanism is used to estimate k:

$$k_i = \begin{cases} max(0, k_{i-1} - 1) & \text{if } \frac{\sum_{j=0}^{lb} loss_j}{lb} > loss_i \\ min(k_{i-1} + 1, k_{max}) & \text{else if } k < k_{max} \end{cases}$$

However given that each batch might have an individual preference on the value of k, the newly determined value k_i is not directly set but instead pushed into a map of arrays containing the history of each batch's k values. The value of k that actually gets used for pruning the coming backwards passes is determined by randomly drawing from the list from the values of k_i observed over last e epochs.

4.3. Assumptions and Limitations

The methods synthesized in this chapter are tailored to neural network architectures used in the computer vision domain, i.e., it is developed to function on architectural components common in this field such as convolutional layers or linear layers (see Section 2.5). Further, it is assumed in the mathematical derivation of the pruning technique targeted at accelerating the forward pass (Section 4.1.2.2) that training is conducted with SGD as (see Section 2.4.2.1) as optimizer, although compatibility with other gradients-based optimization schemes such as Adam (see Section 2.4.2) is likely possible.

5. Performance Model

Due to special purpose sparse hardware (see Section 3.4.1.3) being unavailable as well as only inhomogeneous infrastructure being available during experiments (see Section 6.2.1), I decided to use a performance model to theoretically compute saved flops during training instead of measuring actual runtime speed-up as well as model compression and pruning overhead.

Before training, the performance models computes FLOPs (plural of FLOPs, not to be confused with FLOP/s) $flops^l$ for each layer $l \in L$ of a single forwards pass based on a randomly generated batch of input data using [321]. Then, during training, the performance model evaluates sparsity sp^l_j of layer l at batch j at the end of the batch and uses this information to estimate sparsity-adjusted FLOPs incurred during the forward and backwards pass of batch j. For computing sparsity-adjusted forward FLOPs $flops^l_{fwd,j}$ of layer l at batch j, the performance model uses

$$flops^l_{fwd,j} = flops^l \cdot (1 - sp^l_j) \tag{5.1}$$

Sparsity adjusted backwards FLOPs $flops^l_{bwd,j}$ of layer l at batch j are computed based on the simplifying assumption that a backwards pass incurs twice as many FLOPs as a forwards pass (error backpropagation plus weight update step, an assumption also used in [23] and illustrated in Figure 5.1) but takes into account gradient sparsity $sp^l_{grad,j}$ and gradient accumulation ac (i.e. number of gradients which are accumulated before a weight update is executed, see Section 4.2.2).

$$flops^l_{bwd,j} = \frac{2 \cdot flops^l_{fwd,j} \cdot (1 - sp^l_{grad,j})}{ac} \tag{5.2}$$

Multiplying the sparsity-adjusted value $2 \cdot flops^l_{fwd,j}$ for forward FLOPs with $(1 - sp^l_{grad,j})$ and dividing by ac, which is also used in a similar fashion by [23] for backwards-pass FLOPs estimation, is based on the complexity-reducing premise that the update of a sparse weight with a sparse gradient incurs FLOPs proportional to the fraction of non-zero elements present in both the gradient tensor as well as the weights tensor and that, due to gradient accumulation ac limiting the frequency of such weights updates, these FLOPs are only incurred every ac batches. The total amount of FLOPs incurred during training $flops_{total}$ is then given by the sum of all sparsity-adjusted forward and backwards FLOPs $flops^l_{fwd,j}$ and $flops^l_{bwd,j}$ recorded for all layers l and batches j, i.e.

$$flops_{total} = \sum_{j}^{n} \sum_{l}^{L} (flops^l_{fwd,j} + flops^l_{bwd,j})$$

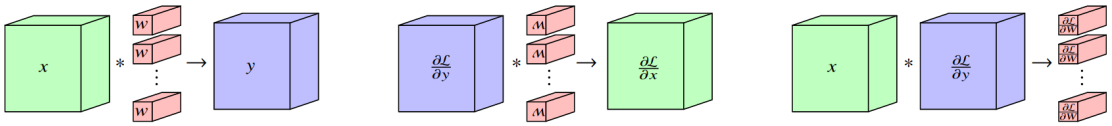


Figure 5.1.: Exemplary illustration of the three major operations occurring in a convolutional layer during training. Left to right: forward pass ($\mathbf{x} * \mathbf{W} \rightarrow \mathbf{y}$), error backpropagation ($\frac{\partial \mathcal{L}}{\partial \mathbf{y}} * \mathbf{W}^{\circlearrowleft} \rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}}$) and weight update ($\mathbf{x} * \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$). The training of linear or fully connected layers is similar but multiplication and $\mathbf{W}^T$ are used instead of convolution [11] and $\mathbf{W}^{\circlearrowleft}$ in the backpropagation. $\mathcal{L}$ denotes loss, $\mathbf{x}$ denotes input activations, $\mathbf{y}$ denotes output activations, $\mathbf{W}$ denotes weights, $\circlearrowleft$ denotes 180° filter-wise rotation.

where $n =$ batches in dataset $\cdot$ epochs. Estimated training speed-up SU is then given by

$$SU = \frac{flops_{total}^{dense}}{flops_{total}^{sparse} + oh}$$

whereby $flops_{total}^{dense}$ is obtained under the assumption that Equations (5.1) and (5.2) are evaluated using $ac = sp_{grad,j}^{l} = sp_{j}^{l} = 1 \forall j, l$ while for $flops_{total}^{sparse}$, the empirically measured values are used. Pruning overhead oh is computed as the sum of the selected combination of pruning heuristics and described in detail below.

5.1. Overhead

Overhead is, in principle, accounted for dependent on the combination of approaches (see Section 4.2) that is employed in a specific experiment for all layers l and batches j i.e.

$$oh = \sum_{m \in M} \sum_{j}^{n} \sum_{l}^{L} oh_{m,j}^{l} \cdot \mathbf{1}(m)$$

with $M \subseteq \{ac, search, mask, topk, admm, app\}$ denoting the set of different overhead types present in a given experiment and $\mathbf{1}$ denoting the indicator function.

5.1.1. Gradient Accumulation

For gradient accumulation (see Section 4.2.2) with number of elements n^l of layer l at batch j, overhead is computed via:

$$oh_{ac,j}^{l} = n^{l} \cdot (1 - sp_{grad,j}^{l})$$

This estimation of gradient accumulation is motivated by the accumulation of two gradients requiring the element-wise summation of two tensors, namely the accumulated gradient stored for the next weight update an the newly obtained gradient at batch j. Taking into account sparsity, the overhead of adding the new gradient to the accumulated gradients is adjusted for the ratio of non-zero elements of the gradient that is added.

5.1.2. RE-Pruning

For RE-Pruning's search algorithm (see Section 4.1.2.3) with density $d_j^l = (1 - sp_j^l)$, layer dimensions D, scaling factor σ_m, number of samples ζ_m and $m \in c, l$, lookback lb, overhead for layer l at batch j is given by:

$$oh_{search,j}^l = \frac{2 \cdot d_j^l \cdot (n^l + \zeta_m \cdot \sigma_m \cdot \prod_{k=0}^{|D|} D_k)}{lb}$$

First, the components of the numerator of equation (4.16) is computed using the accumulated gradient $\sum_{k=j-lb}^{j} \nabla_{\boldsymbol{W}_k}^l$ to obtain the weights as they were lb batches in the past, $\boldsymbol{W}_{(j+1)-lb}^l$, requiring $2 \cdot n^l$ operations due to multiplication with learning rate η and element wise addition with $\boldsymbol{W}_{(j+1)}^l$, adjusted for layer density d_j^l. Then based on the intra layer formulation of Equation (4.16), for each of the ζ_m samples of scale σ_m, the search heuristic computes the norm twice (once for each sample of $\boldsymbol{W}_{(j+1)}^l$ in the denominator and once for each sample of $\boldsymbol{W}_{(j+1)-lb}^l$ in the numerator), incurring $d_j^l \cdot \zeta_m \cdot \sigma_m \cdot 2 \cdot \prod_{k=0}^{|D|} D_k$ operations adjusted for density in total. The term is divided by lb because these operations are only executed every lb batches. Because, as described in Section 4.1.2.3, the scaling parameters only represent and upper bound for the size of the actual samples used in the computation, the overhead estimation here represents an upper bound as well and the true overhead is generally smaller. Mask computation of RE-Pruning with quantile κ_m and number of samples ζ_m are given by:

$$oh_{mask,j}^l = \frac{n^l \cdot \kappa_m \cdot \zeta_m}{lb}$$

This represents that the mask at batch j with n^l elements is element-wise multiplicatively composed of the κ_m quantile of ζ_m samples. The size of the sample σ_m is not considered in above equation because the binary pruning mask is always of the same size as the weights of the layer that is to be pruned. Again the term is divided by lb because these mask computation is only executed every lb batches. Mask application is given by

$$oh_{app,j}^l = 2 \cdot n^l$$

which considers the element-wise multiplication of the layers weights and gradients with the pruning mask, costing n^l operations each. Because gradients are not always masked however (see Section 4.1.2.4 for the conditions under which they are), above computation of mask application overhead is to be considered an upper bound.

5.1.3. Top-K Gradients

With density d_j^l, layer dimensions D and lookback lb, the Top-K gradients pruning heuristics (see Section 4.1.1.5) overhead for layer l at batch j is given by:

$$oh_{topk,j}^l = \frac{d_j^l \cdot \prod_{k=0}^{|D|} D_k}{lb}$$

The computation of this overhead is motivated by the norm computation of the accumulated gradient in Equation (4.4) adjusted by the density of the layers weights. The density of the layers weights is used as estimator of the density of the accumulated gradient based on the assumption that an element of the a weight at batch j is likely only then zero if it has not been updated by the accumulated gradient of the last lb batches unless the model has converged to a local optimum and the gradient is oscillating. Because these operations are only executed every lb batches, the term is divided by lb. In the MC case (see Section 4.1.1.6), overhead is only computed during sampling epochs. Drawing from as well as updating the probabilities in the categorical distribution is not accounted for due to its costs being negligible. Masking is not accounted for since whole gradients are masked, which in an efficient implementation, can be done by a simple switch and does not involve computationally expensive element-wise multiplications of tensors or similar operations.

5.1.4. ADMM

The overhead incurred by ADMM (see Section 4.2.1) consists of two components: the loss computation dependent on the dual variables as well as the involved norm computations (see [18] for algorithmic details), which is given below for layer l at batch j, as well as the mask application overhead, which is identical to the mask application overhead incurred by RE-Pruning.

$$oh^l_{admm,j} = 3 \cdot n^l + 2 \cdot \prod_{k=0}^{|D|} D_k$$

The update of the dual variables is neglected for simplicity because the dual variables are only updated once at the end of each epoch and hence their updates impact on ADMMs total algorithmic overhead can be considered to be insignificant compared to the overhead that occurs at every batch of ADMM training in the loss computation as well as the mask application.

5.2. Assumptions and Limitations

Besides the assumptions already stated in above description of the performance model, the performance model is as a necessary simplification hardware agnostic and assumes that hardware can optimally exploit the induced sparsity. This necessity of hardware agnosticism stems not only from the practical consideration of the underlying hardware used in the experimental evaluation of this thesis being diverse (see Section 6.2.1), but also from the proposed pruning approach not being dedicated to any particular type of specialized sparse hardware architecture. Regarding the simplifying assumption of optimal sparsity exploitation, this assumption is based on the preliminary observation that works using specialized sparse hardware or simulations of such hardware designed to function optimally with their pruning approach (e.g. [11, 227]) report data indicating exploitation rates (i.e. speedup divided by sparsity-induced network compression) of close

to 100% in certain cases. However a dependency of the exploitation rate on the model that is trained can be observed as well such that the performance model used in this thesis not incorporating this dependency is certainly one of its shortcomings that has to be addressed in future work.

6. Empirical Evaluation

Only certain parts of this thesis could be shown to be advantageous analytically (see 4.1.1.3). Hence, empirical evaluation is required. In this section, I first describe implementation, setup and methodology (hardware, competitors and experiments selection) of this evaluation, conduct an ablation study, analyse my approach's sensitivity to hyperparameters, examine its convergence behaviour and induction of sparsity, illustrate performance improvements and provide a comparison to competing State-of-the-Art works.

6.1. Implementation

The implementation of the approaches synthesized in Section 4 was realized using PyTorch [56] machine learning framework and JetBrains PyCharm Professional IDE [322]. The decision for using PyTorch over other major machine learning frameworks such as TensorFlow [57] was consciously made due to PyTorch's dynamic computation graph making modifications of the networks during training much easier compared to TensorFlows static computation graph model. Version control was realized via the IDEs GitHub [323] integration and publication of the repository is planned after the end of my thesis project.

The implementation adheres to the best of the author's ability to the internal and external quality characteristics described in [324], with a particular focus on the external characteristics correctness, efficiency and adaptability and the internal characteristics flexibility, readability and testability because a focus on these characteristics will be most advantageous when adapting the implementation for different data sets, tasks and models in future work while at the same time guaranteeing reliable and actionable results. This is achieved by the use of defensive programming [324], adherence to classic OOP principles [324] as well as the application of design and architecture patterns [325, 326] and general best practices (documentation, style guides according to PEP8 [327]) where seen fit.

6.2. Setup

6.2.1. Hardware

Experimental evaluation was conducted on 3 different machines configured as listed in Table 6.1. Large-scale experiments were exclusively conducted on machine 0 (DGX-1 [328], illustration of a Tesla V100 as used in DGX-1, see Figure 6.1) provided by Visualization and Data Analysis (VDA) research group. Machine 0 had job scheduling via slurm [329] and I was given an allowance of resources as described in table 6.1. I split the available

ID	GPUs	GPU MEM (each)	CPU Cores	MEM (total)
0	2x Tesla V100	16GB	16x E5-2698 v4[1]	64 GB
1	1x Tesla P100	12GB	64x Gold 6130[1]	252 GB
2	1x GTX 1650 Ti[3]	4GB	8x i7-10750H[2]	46 GB

Table 6.1.: Hardware used for experimental evaluation, [1]Intel Xeon, [2]Intel Core, [3]GeForce

resources on machine 0 to be able to run two series of experiments in parallel. Machine 1, which was provided by the Theory and Application of Algorithms (TAA) research group, on the condition that I do not interfere with experiments of the providing research groups other researchers. Machine 1 had no job scheduling and hence could only be used occasionally in times where usage by others were low. Machine 2 was my laptop and, due to its limited hardware capabilities, mainly used for prototyping during development, small-scale ablation studies and evaluation of results (using Jupyter Notebook [330]) obtained from more powerful machines 1 and 2.

Figure 6.1.: . Illustration of Tesla V100 as used in DGX-1 [12]

6.2.2. Methodology

As discussed in Section 4.2, several different combinations of the proposed approaches are possible. In order to investigate the effects produced by these different combinations as well as their sensitivity to hyperparameters, I will first conduct an ablation study using relative small and easily trainable combinations of CNN architectures and datasets. The most promising candidate methods are then selected for larger scale evaluation on scientifically competitive architecture-dataset combinations and compared against other State-of-the-Art solutions by carefully selected metrics.

6.2.3. Selection of Primary and Secondary Competitors

Considered as primary competitors are all works listed in Tables 3.9 and 3.10 because these works were designed to accelerate training through pruning and also document such and acceleration during training. Works listed in Tables 3.7 and 3.8 will not be treated as primary competitors, but only as "nice to have if I beat them" (i.e. secondary competitors) because, although these works could have a potential impact on training acceleration (see discussion in Section 3.4.1.4), no such acceleration was documented by

the original authors. Other works listed in Section 3.4 which are not having at least a potential advantage during training (e.g. pruning applied very late during training or extensive retraining required s.t. overall training effort is likely higher than regular training) are ignored here for the time and space constraints of this thesis.

6.2.4. Selection of Experiments

My selection of experiments is primarily motivated by competing State-of-the-Art works on accelerating training through intra-training pruning listed in Tables 3.9 and 3.10 as these are the most important competitors to the approaches designed in this thesis. Secondly, I optionally also conduct experiments comparable with works listed in Tables 3.7 and 3.8 which are focused on inference acceleration, though producing advantages during inference, despite desirable, is not the main goal of my work. Thirdly, my selection of experiments was motivated by the goal to cover as many architectural categories (see Section 2.5.1) as possible within the time frame of this thesis to evaluate the generalizability of my proposed approaches.

Due to the large number of competitors and different setups in terms of epochs, optimizers and learning rates for different architectures and datasets, leading to different outcomes in terms of convergence and final test accuracy, I chose to use a standardized training approach for all my experiments and use fixed epochs and learning rates across all experiments and only track final test accuracy delta instead of final test accuracy to help with comparability.

6.2.5. Selection of Measured Metrics

Measured metrics are from the categories Quality, Acceleration, Sparsity and Compression (see Table 3.2 for overview) and are motivated by competing works in Tables 3.9 and 3.10 of as well as the general metrics preferences in the field seen in Table 3.4 As mentioned previously, quality is measured by ΔAccuracy = Baseline Accuracy − Pruned Accuracy where accuracy refers to test accuracy, acceleration is measured as relative reduction of FLOPs during inference $\text{FLOPs(I)} = 1 - \frac{\text{Pruned FLOPs(I)}}{\text{Baseline FLOPs(I)}}$ and $\text{FLOPs(T)} = 1 - \frac{\text{Pruned FLOPs(T)}}{\text{Baseline FLOPs(T)}}$, sparsity and compression of parameters P. (i.e. all of the models trainable weights) and channels/filters C/F. (only found in convolutional layers, see also Figure 2.10) are measured by the $P = 1 - \frac{PrunedParameters}{TotalParameters}$ and $C/F = 1 - \frac{PrunedChannels/Filters}{TotalChannels/Filters}$.

6.3. Results and Discussion

6.3.1. Ablation Study

To explore the performance of different combinations of approaches, an ablation study was conducted using the LeNet-5/MNIST, AlexNet-S/CIFAR-10 and ResNet-18/CIFAR-10 model/dataset combinations. Examples of the MNIST and CIFAR-10 datasets are illustrated in Figures 6.2 and 6.3. These combinations were chosen motivated by Table 3.3

Figure 6.2.: . Examples of handwritten digits in the MNIST dataset [2, p. 46]

as well as because a large number of experiments can be conducted quickly with these combinations. While AlexNet-S/CIFAR-10 and ResNet-18/CIFAR-10 were used solely for evaluating different combinations of algorithms, LeNet-5/MNIST was also used to evaluate the different algorithms' sensitivity to certain hyperparameter choices. As baseline, AlexNet-S/CIFAR-10 and ResNet-18/CIFAR-10 were trained for 100 epochs with SGD, batch-size 256 and learning rate lr=0.1 (also denoted as η in Sections 2.4.2.1 and 2.4.2.2), LeNet-5/MNIST used the same settings but was only trained for 10 epochs. Multistep learning rate scheduling was used for SGD with steps at 50% and 70% of total training epochs and gamma, the factor by which learning rate is reduced at each scheduled step, set to 0.1. Experiments using REP were conducted using the same setting due to REP being designed around SGD. Experiments using GDTopK and GDTopKMC as well as ADMMI and AMMR were conducted using Adam and lr=0.001 and no learning rate scheduling due to Adam being used in the orignal ADMM paper and implementation [18]. Because SGD generalizes better on computer vision tasks than Adam (see Section 2.4.2.2), baselines obtained via SGD training can still be considered valid or even harder to beat in terms of generalization, so the found accuracy reductions in these cases can be considered upper bounds. Experiments using any combination of GDTopK, GDTopKMC, ADMMI, AMMR with REP use Adam. These settings were also used during other experiments of the ablation study. Regarding the specific hyperparameters of the intra-training pruning algorithms, please refer to the GitHub repository[1] associated with this thesis for details.

[1]Link: `https://github.com/lorenz0890/multi-directional-pruning`.

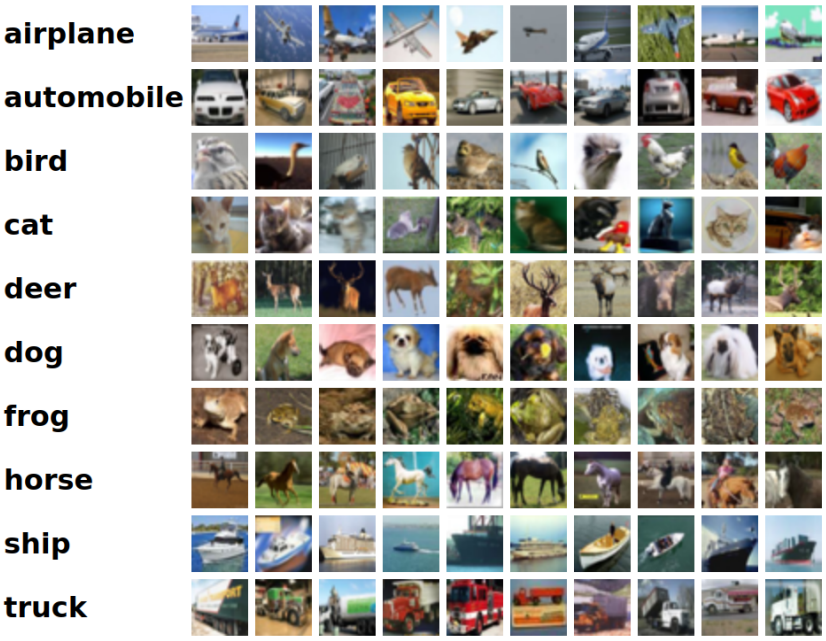


Figure 6.3.: . Examples of images and associated classes in the CIFAR-10 dataset [13]

6.3.1.1. Combinations of Approaches

Tables 6.2, 6.3 and 6.4 show the results of the ablation study and illustrate the contribution to the selected metrics of different components of complex combined approaches such as REP+GDTopKMC+AC+DK+ADMMI and REP+GDTopKMC+AC+DK+ADMMR. In the first paragraph of this section, observations regarding differences in the performance of the tested combinations on different architecture/dataset combinations are discussed. In the next three paragraphs, the performance of the three individual components (REP, GDTopK, ADMM) of the combined approaches as well as and their variants is explored. Finally in the last paragraph, it is concluded that based on the observations of the ablation study, REP+GDTopKMC+AC+DK+ADMMI provides the best trade-off between acceleration and accuracy degradation across all architecture/dataset combinations assessed.

Different Results on Different Architecture/Dataset Combinations For LeNet-5/MNIST, the best Δ Acc is achieved by ADMMI with an increase of 0.02 percentage points (p.p.) at 30.97% reduction in training FLOPs while the highest reduction of FLOPs during training is reached by REP+AC which reduces training FLOPs by 95.67% at a Δ Acc of -0.94 p.p. For the combined approaches REP+GDTopKMC+AC+DK+ADMMI and REP+GDTopKMC+AC+DK+ADMMR, -0.54 p.p. and 83.82% FLOPs(T) as well as -0.87 p.p. and 78.4% FLOPs(T) are observed respectively. For AlexNet-s/CIFAR-10, the smallest accuracy degradation is achieved by GDTopKMC+AC+DK+ADMMR at -0.12 percentage points (p.p.) at 40.6% FLOPs(T) while highest FLOPs(T) is reached by REP+GDTopKMC+AC+DK+ADMMI with 64.57% at -1.9 p.p. Δ Acc

For REP+GDTopKMC+AC+DK+ADMMR, Δ Acc -2.37 p.p. and FLOPs(T) 60.85% was observed. With ResNet-18/CIFAR-10, the best Δ Acc was reached with GDTopKMC+AC+DK at 2.81 p.p. at 31.69 % FLOPs(T) while the best FLOPs(T) was observed for REP+GDTopKMC+AC+DK+ADMMI at 66.95% at -1.42 p.p. degradation. The second complex approach REP+GDTopKMC+AC+DK+ADMMR performed at Δ Acc -4.93 p.p. with 49.91% FLOPs(T).

Performance of GDTopK Variants As expected, GDTopK variants by themselves (i.e., without REP or ADMMI or ADMMR) do not induce a reduction in parameters or have any inference time advantages but only lead to sparsification of gradients. Interestingly though, the GDTopK only variants led to an increase in test accuracy by 1.39 p.p to 2.81 p.p. at gradient sparsity levels from 32.8% to 73.64% on ResNet-18/CIFAR-10, indicating that at least for this architecture/dataset combination, gradients contain large portions of redundant information that can be discarded without degradation of the resulting network.

Performance of REP Variants REP only variants (i.e., without GDTopK or ADMMI/R) surprisingly showed a very mixed performance: while they produced the highest training acceleration by 95.67% FLOPs(T) at neglectable accuracy decrease (-0.94 p.p) for LeNet-5/MNIST at 98.41% pruning rate P(I), similarly high pruning rates (e.g., 91.8% P(I)) led only to very minor training acceleration of 17.81% to 35.65% FLOPs(T) on ResNet-18/CIFAR-10 and at best mediocre results on AlexNet-s/CIFAR-10. Given that the amount of acceleration and network compression produced by pruning is dependent on the layer type (pruning convolutional layers mostly reduced computations while pruning linear layers mostly leads to network compression), I conjecture the observation is caused by different components being pruned in these different architecture/dataset combinations because REP in its current states disregards potential gains in performance or compression as selection criteria.

Performance of ADMM Variants With regard to ADMMR and ADMMI, none of the two approaches performed best in any of the experiments of the ablation study. However, the assumption made in Section 4.2.1 that through re-engineering the originally proposed ADMMR can be modified such that improved training speed-ups are achieved holds: ADMMI outperforms ADMM in terms of training FLOPs reduction FLOPs(T) and Δ Acc in two out of three architecture/dataset combinations (ResNet-18 and AlexNet-s on CIFAR-10) and shows better Δ Acc on LeNet-5/MNIST as well.

The Best Trade-Off The approach that produced desirable results most consistently across all architecture/dataset combinations evaluated during the ablation study turned out to be REP+GDTopKMC+AC+DK+ADMMI, which produced the highest training acceleration in both AlexNet-s and ResNet-18 on CIFAR-10 and represents a good trade-off between accuracy degradation and acceleration in all evaluated architecture dataset combinations. Figures 6.4, A.5 and A.4, which illustrate the relation

Approach	**Δ Acc**	**C/F(I)**	**P(I)**	**FLOPs(I)**	**FLOPs(T)**	**⊘ G(T)**
ADMMI	**+0.02 p.p.**	84.52%	93.26%	80.41%	30.97%	48.5%
ADMMR	−0.01 p.p.	74.49%	90.26%	73.44%	39.97%	56.85%
GDTopK	−0.30 p.p.	0.00%	0.00%	0.00%	51.25%	88.96%
GDTopKMC	−0.15 p.p.	0.00%	0.00%	0.00%	36.8%	76.68%
GDTopKMC+AC	−0.10 p.p.	0.00%	0.00%	0.00%	46.02%	69.5%
GDTopKMC+AC +DK	−0.19 p.p.	0.00%	0.00%	0.00%	26.9%	41.51%
GDTopKMC+AC +DK+ADMMI	−0.27 p.p.	60.75%	70.17%	57.41%	63.50%	75.56%
GDTopKMC+AC +DK+ADMMR	−0.24 p.p.	91.76%	98.59%	91.68%	50.04%	43.53%
REP	−0.34 p.p.	92.50%	21.19%	74.48%	74.31%	11.06%
REP+AC	−0.94 p.p.	**94.75%**	98.41%	91.19%	**95.67%**	59.23%
REP+ADMMI	−0.35 p.p.	91.76%	98.59%	91.68%	79.11%	84.07%
REP+ADMMR	−0.30 p.p.	91.77%	98.60%	91.68%	81.64%	37.64%
REP+AC+ADMMI	−0.40 p.p.	91.76%	98.59%	91.68%	89.26%	87.07%
REP+AC+ADMMR	−0.63 p.p.	91.87%	**98.61%**	**91.75%**	89.7%	**93.01%**
REP+GDTopKMC +AC+DK	−0.51 p.p.	47.56%	71.4%	47.65%	74.23%	21.38%
REP+GDTopKMC +AC+DK+ADMMI	−0.54 p.p.	91.77%	98.6%	91.68%	83.82%	87.27%
REP+GDTopKMC +AC+DK+ADMMR	−0.87 p.p.	91.80%	98.60%	91.70%	78.40%	41.21%

Table 6.2.: Ablation study LeNet-5/MNIST. Fields with % read as reduced by X%. FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F(I) = convolution kernels or filters at inference time (i.e. after training has finished), P(I) = parameters at inference time, ⊘ G(T) = average gradient sparsity during training.

Approach	**Δ Acc**	**C/F(I)**	**P(I)**	**FLOPs(I)**	**FLOPs(T)**	**⊘ G(T)**
ADMMI	−1.91 p.p.	93.81%	98.45%	46.12%	51.44%	86.69%
ADMMR	−3.41 p.p.	93.81%	98.45%	46.12%	34.79%	61.27%
GDTopK	−0.86 p.p.	0.00%	0.00%	0.00%	32.48%	90.16%
GDTopKMC	−1.49 p.p.	0.00%	0.00%	0.00%	32.14%	88.70%
GDTopKMC+AC	−3.01 p.p.	0.00%	0.00%	0.00%	45.40%	89.06%
GDTopKMC+AC +DK	−0.55 p.p.	0.00%	0.00%	0.00%	32.01%	27.45%
GDTopKMC+AC +DK+ADMMI	−0.69 p.p.	93.81%	98.45%	46.12%	62.81%	87.98%
GDTopKMC+AC +DK+ADMMR	**−0.12 p.p.**	87.19%	98.61%	40.60%	40.83%	49.49%
REP	−4.94 p.p.	75.64%	42.06%	25.05%	21.46%	0.30%
REP+AC	−6.64 p.p.	89.47%	20.39%	25.75%	45.20%	2.67%
REP+ADMMI	−3.38 p.p.	93.83%	98.45%	46.12%	60.98%	**92.21%**
REP+ADMMR	−5.15 p.p.	**94.19%**	**98.52%**	**46.24%**	49.74%	65.03%
REP+AC+ADMMI	−3.09 p.p.	93.81%	98.45%	46.12%	63.95%	91.84%
REP+AC+ADMMR	−7.67 p.p.	94.07%	98.51%	46.22%	58.67%	32.68%
REP+GDTopKMC +AC+DK	−8.29 p.p.	72.56%	79.13%	25.70%	47.38%	76.36%
REP+GDTopKMC +AC+DK+ADMMI	−1.90 p.p.	93.81%	98.45%	46.12%	**64.57%**	91.78%
REP+GDTopKMC +AC+DK+ADMMR	−2.37 p.p.	93.99%	98.48%	46.17%	60.85%	74.89%

Table 6.3.: Ablation study AlexNet-S/CIFAR-10. Fields with % read as reduced by X%. FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F(I) = convolution kernels or filters at inference time (i.e. after training has finished), P(I) = parameters at inference time, ⊘ G(T) = average gradient sparsity during training.

between Δ Acc and FLOPs(T) further support this finding as they consistently show REP+GDTopKMC+AC+DK+ADMMI in the top-right corner across all three evaluated architecture/dataset combinations. It can further be seen that REP+GDTopKMC+AC+DK+ADMMI trained models have a very accuracy density (accuracy per network density) and REP+GDTopKMC+AC+DK+ADMMI has negligible algorithmic overhead. Consequently, only REP+GDTopKMC+AC+DK+ADMMI will be used for further evaluation in this thesis.

Approach	**Δ Acc**	**C/F(I)**	**P(I)**	**FLOPs(I)**	**FLOPs(T)**	**⊘ G(T)**
ADMMI	+0.32 p.p.	70.53%	70.54%	46.10%	59.75%	**91.92%**
ADMMR	−4.72 p.p.	70.53%	70.54%	46.10%	24.1%	64.53%
GDTopK	+1.39 p.p.	0.00%	0.00%	0.00%	9.21%	73.64%
GDTopKMC	+2.29 p.p.	0.00%	0.00%	0.00%	7.22%	70.36%
GDTopKMC+AC	+1.98 p.p.	0.00%	0.00%	0.00%	33.64%	73.36%
GDTopKMC+AC +DK	**+2.81 p.p.**	0.00%	0.00%	0.00%	31.69%	32.80%
GDTopKMC+AC +DK+ADMMI	+0.46 p.p.	70.53%	70.54%	46.10%	65.80%	90.84%
GDTopKMC+AC +DK+ADMMR	−21.52 p.p.	94.28%	94.27%	**61.53%**	37.75%	48.27%
REP	+1.26 p.p.	**97.28%**	**95.69%**	21.02%	17.81%	2.53%
REP+AC	−0.06 p.p.	93.34%	91.8%	16.95%	35.65%	56.65%
REP+ADMMI	+0.15 p.p.	89.49%	89.18%	47.65%	57.93%	87.65%
REP+ADMMR	−4.64 p.p.	88.56%	88.26%	47.54%	27.27%	57.02%
REP+AC+ADMMI	+0.30 p.p.	87.36%	87.08%	47.36%	66.26%	88.07%
REP+AC+ADMMR	−7.29 p.p.	86.93%	86.66%	47.35%	45.24%	75.45%
REP+GDTopKMC +AC+DK	−0.06 p.p.	92.35%	90.83%	15.71%	35.24%	73.69%
REP+GDTopKMC +AC+DK+ADMMI	−1.42 p.p.	90.48%	90.15%	47.72%	**66.95%**	91.63%
REP+GDTopKMC +AC+DK+ADMMR	−4.93 p.p.	90.9%	90.57%	49.40%	49.91%	83.41%

Table 6.4.: Ablation study ResNet-18/CIFAR-10. Fields with % read as reduced by X%. FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F(I) = convolution kernels or filters at inference time (i.e. after training has finished), P(I) = parameters at inference time, ⊘ G(T) = average gradient sparsity during training.

6.3.1.2. Hyper Parameter Sensitivity

The hyperparameter sensitivity study of the approach found in the previous section to be most promising, REP+GDTopKMC+AC+DK+ADMMI, was conducted on the LeNet-5/MNIST architecture/dataset combination because its simplicity allowed vast numbers of different combinations to be evaluated within the time frame of this thesis. In total, 255 different hyperparameter combinations were evaluated for LeNet-5/MNIST, and the individual parameters ranges are given by Table 6.5. The sensitivity of REP+GDTopKMC+AC+DK+ADMMI to different hyperparameters is visualized by Figure 6.5. As illustrated, test accuracy most strongly affected by the number of pruning epochs, i.e., the number of epochs that

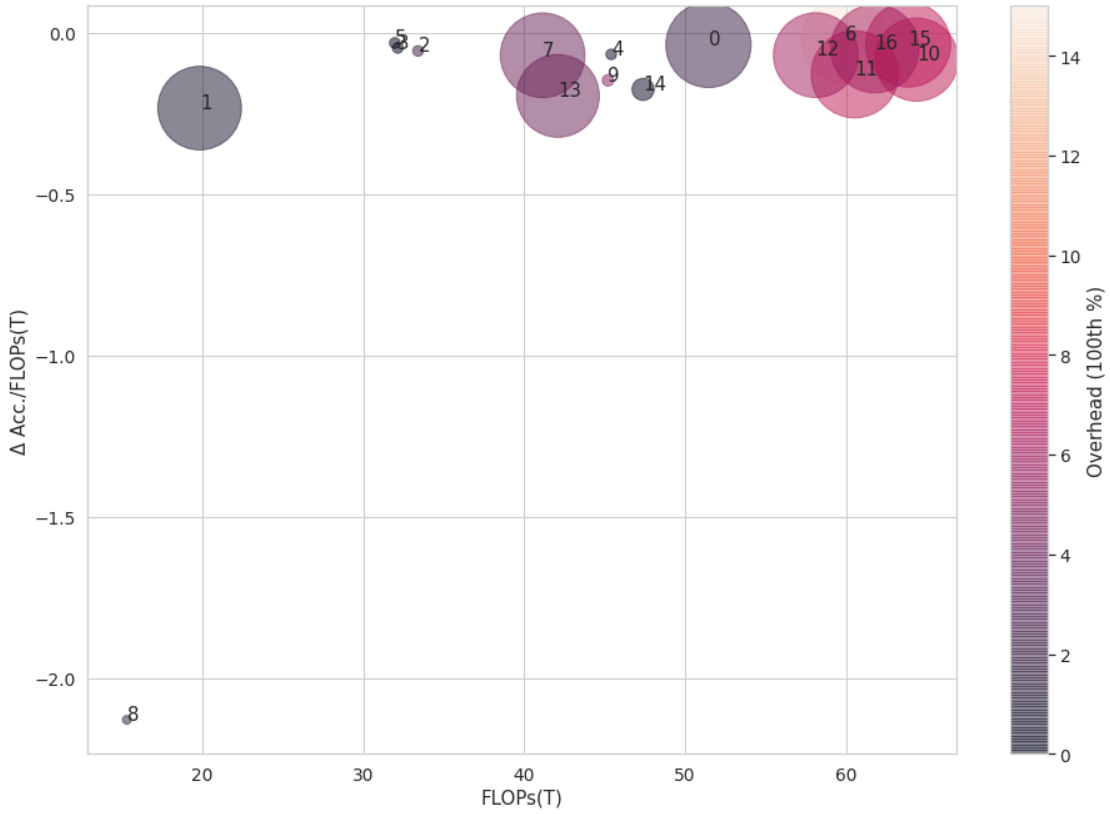


Figure 6.4.: Ball scatter plot of results of Table 6.3 for AlexNet-S/CIFAR-10, 0=ADMMI, 1=ADMMR, 2=GDTopK, 3=GDTopKMC, 4=GDTopKMC+AC, 5=GDTopKMC+AC+DK, 6=GDTopKMC+AC+DK+ADMMI, 7=GDTopKMC+AC+DK+ADMMR, 8=REP, 9=REP+AC, 10=REP+AC+ADMMI, 11=REP+AC+ADMMR, 12=REP+ADMMI, 13=REP+ADMMR, 14=REP+GDTopKMC+AC+DK, 15=REP+GDTopKMC+AC+DK+ADMMI, 16=REP+GDTopKMC+AC+DK+ADMMR. Ball size encodes accuracy density (test accuracy per network density). Method overhead measured in hundredth % (denoted as 100th % in the figure) of total training FLOPs.

are used to generate the pruning mask used by the REP component of the algorithm. Since the observed connection is an anti-correlation, it can be concluded that the lower the number of pruning epochs, the higher the final test accuracy is, which fits nicely into the concept of eager pruning as intended use of REP (see Section 4.1.2.4). Measured speed-ups (i.e., reductions in FLOPs) are most strongly affected the softness factor that is applied during the pruning epochs of REP which controls how "hard" the network is pruned: the higher the softness value is chosen, the lower the speed-up, which can be intuitively explained by high softness values leading to fewer parameters of the network

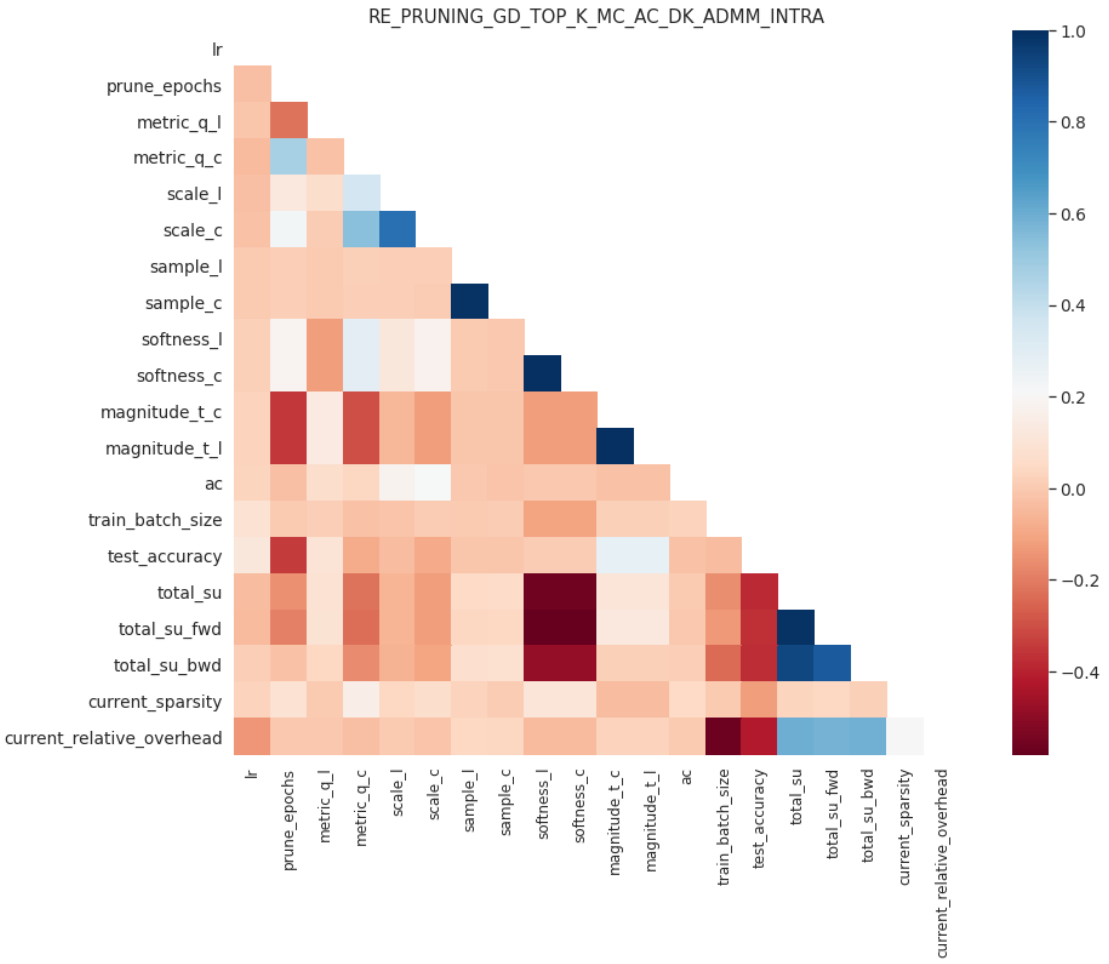


Figure 6.5.: Correlation matrix for REP+GDTopKMC+AC+DK+ADMMI. Please refer to Sections 4.1.1.6, 4.1.2.4 and 4.2 for details on individual parameters.

being pruned. Interestingly, this effect also could be observed for test accuracy, although the anti-correlation is less strong than it is with speed up, which is counterintuitive due to the softness factor being introduced with the intention to avoid network degradation due to overly hard pruning, hence a more thorough investigation across multiple architectures and datasets (e.g., AlexNet, ResNet-18 on CIFAR-10/100) should be conducted. The strong dependency of algorithmic overhead on batch size which could be observed is most likely the result of the chosen combinations of hyperparameters: while the lookback parameter was kept constant at a value of 50 across all experiments with LeNet-5/MNIST during the hyperparameter sensitivity study, batch size was varied between 64 and 256; hence smaller batch sizes led to more frequent executions of REP and GDTopKMC algorithms. For other approaches and combinations of approaches, correlation matrices can be found in Figure A.6 to Figure A.21 and respective hyper parameter ranges in

Parameter	Count	Mean	Std	Min	Max
lr	$2.55 \cdot 10^2$	$1.08 \cdot 10^{-3}$	$9.86 \cdot 10^{-4}$	$1.00 \cdot 10^{-4}$	$1.00 \cdot 10^{-2}$
prune_epochs	$2.55 \cdot 10^2$	$1.84 \cdot 10^{-1}$	$5.55 \cdot 10^{-1}$	0.00	2.00
metric_q_l	$2.55 \cdot 10^2$	$5.12 \cdot 10^{-2}$	$1.29 \cdot 10^{-2}$	$2.50 \cdot 10^{-2}$	$1.00 \cdot 10^{-1}$
metric_q_c	$2.55 \cdot 10^2$	$8.45 \cdot 10^{-2}$	$6.91 \cdot 10^{-2}$	$2.50 \cdot 10^{-2}$	$2.50 \cdot 10^{-1}$
scale_l	$2.55 \cdot 10^2$	$1.15 \cdot 10^{-1}$	$4.47 \cdot 10^{-2}$	$1.00 \cdot 10^{-1}$	$2.50 \cdot 10^{-1}$
scale_c	$2.55 \cdot 10^2$	$1.54 \cdot 10^{-2}$	$1.99 \cdot 10^{-2}$	$1.00 \cdot 10^{-2}$	$1.00 \cdot 10^{-1}$
sample_l	$2.55 \cdot 10^2$	$9.89 \cdot 10^2$	$2.53 \cdot 10^2$	$1.00 \cdot 10^2$	$2.00 \cdot 10^3$
sample_c	$2.55 \cdot 10^2$	$9.91 \cdot 10^2$	$2.51 \cdot 10^2$	$1.00 \cdot 10^2$	$2.00 \cdot 10^3$
softness_l	$2.55 \cdot 10^2$	$5.66 \cdot 10^{-1}$	$4.30 \cdot 10^{-1}$	0.00	$9.00 \cdot 10^{-1}$
softness_c	$2.55 \cdot 10^2$	$5.66 \cdot 10^{-1}$	$4.30 \cdot 10^{-1}$	0.00	$9.00 \cdot 10^{-1}$
magnitude_t_c	$2.55 \cdot 10^2$	$9.29 \cdot 10^{-4}$	$2.42 \cdot 10^{-4}$	$1.00 \cdot 10^{-4}$	$1.00 \cdot 10^{-3}$
magnitude_t_l	$2.55 \cdot 10^2$	$9.29 \cdot 10^{-4}$	$2.42 \cdot 10^{-4}$	$1.00 \cdot 10^{-4}$	$1.00 \cdot 10^{-3}$
ac	$2.55 \cdot 10^2$	6.05	2.11	4.00	$1.20 \cdot 10^1$
train_batch_size	$2.55 \cdot 10^2$	$1.77 \cdot 10^2$	$1.03 \cdot 10^2$	$6.40 \cdot 10^1$	$5.12 \cdot 10^2$

Table 6.5.: Hyperparamater range for REP+GDTopKMC+AC+DK+ADMMI. Please refer to Sections 4.1.1.6, 4.1.2.4 and 4.2 for details on individual parameters.

Tables A.1 to A.16.

6.3.2. Convergence and Induction of Sparsity

6.3.2.1. Convergence Analysis

An interesting perspective on approaches that prune, quantize or otherwise perturb the training of a neural network is convergence. Hence, for the approach I recognized in Section 6.3.1 as generalizing best across architectures and datasets, REP+GDTopKMC+AC+DK+ADMMI, the convergence on the two benchmark architecture/dataset combinations AlexNet-s/CIFAR-10 and ResNet18/CIFAR-10 is evaluated. In both cases, it can be observed that training convergence is slightly hampered compared to regular training (Figures A.1 and 6.6), which is not surprising given the aggressive stance REP+GDTopKMC+AC+DK+ADMMI takes on pruning of gradients and weights. Interestingly though, as illustrated by Figures A.2 and 6.7, with REP+GDTopKMC+AC+DK+ADMMI generalization is slightly improved. I conjecture this can be attributed to the regularizing effect of network perturbations as documented by e.g. [331]. With regard to the induction of sparsity by REP+GDTopKMC+AC+DK+ADMMI, Figure 6.8 andA.3 illustrate the sparsification through REP of the network very early during training as well the continued adaption through repeated application of ADMM via ADMMI. Another perspective on sparsity induction is to look at it in a layer-wise manner. As can be seen by Figure 6.9, REP+GDTopKMC+AC+DK+ADMMI does not induce sparsity uniformly across the network but has clear layer preferences, illustrating its ability to chose layers for pruning which are less crucial to the training's learning effort: large

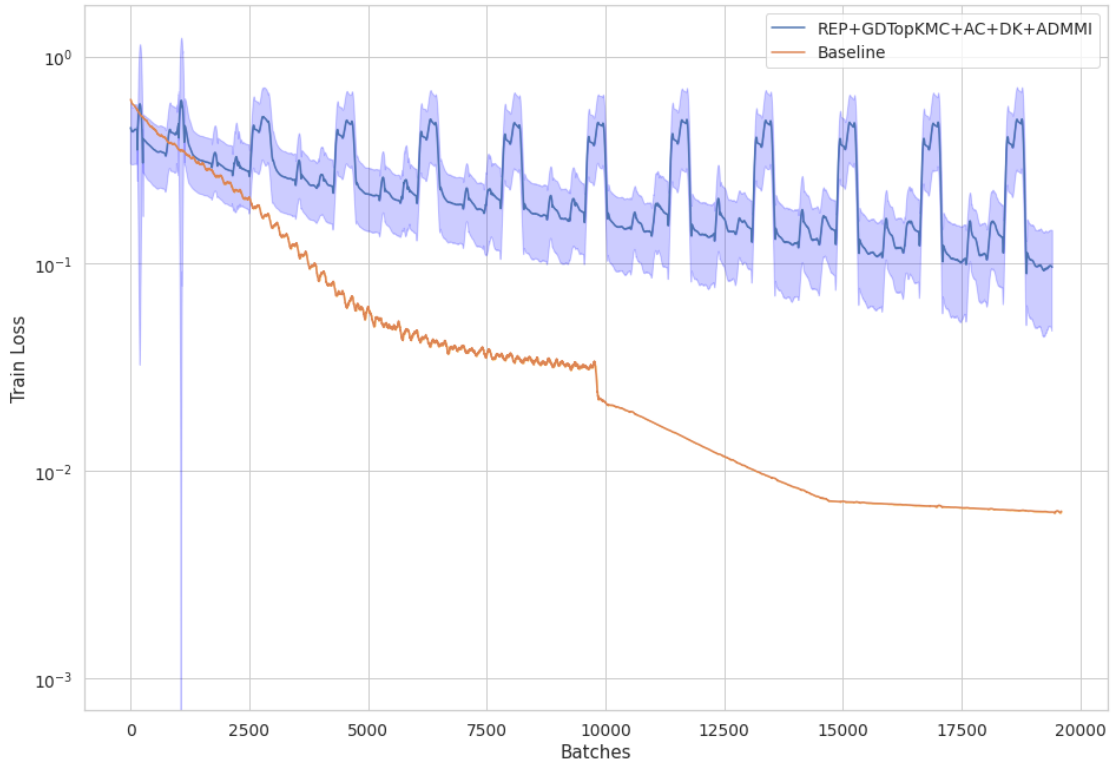


Figure 6.6.: REP+GDTopKMC+AC+DK+ADMMI train loss (explanation see Section 2.4.4) on AlexNet-s/CIFAR-10, 11 runs, mean and standard deviation, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability.

top-level layers get pruned stronger than smaller bottom layers close to the output layer where smaller perturbations might have a greater derogatory impact on network accuracy and learned information is more densely stored. With regard to gradients, a similar but less pronounced effect can be observed in Figure 6.10. Which gradients are deleted by REP+GDTopKMC+AC+DK+ADMMI is indicated by sections with 100% gradient sparsity.

6.3.2.2. Structure of Induced Sparsity

The structure of the induced sparsity is visualized in Figures 6.11 and 6.12 as well as A.22 and A.23. As can be seen, large numbers of filters have been zeroed out, particularly in the second convolutional layer of LeNet-5 (Figure 6.11), reducing computational cost in this otherwise expensive type of layer. Additionally, it should be noted that large portions of pruned filters are adjunct to each other, which illustrates that REP+GDTopKMC+AC+DK+ADMMIs capability to prune convolutional layers in a structured manner even above filter level which could potentially have positive

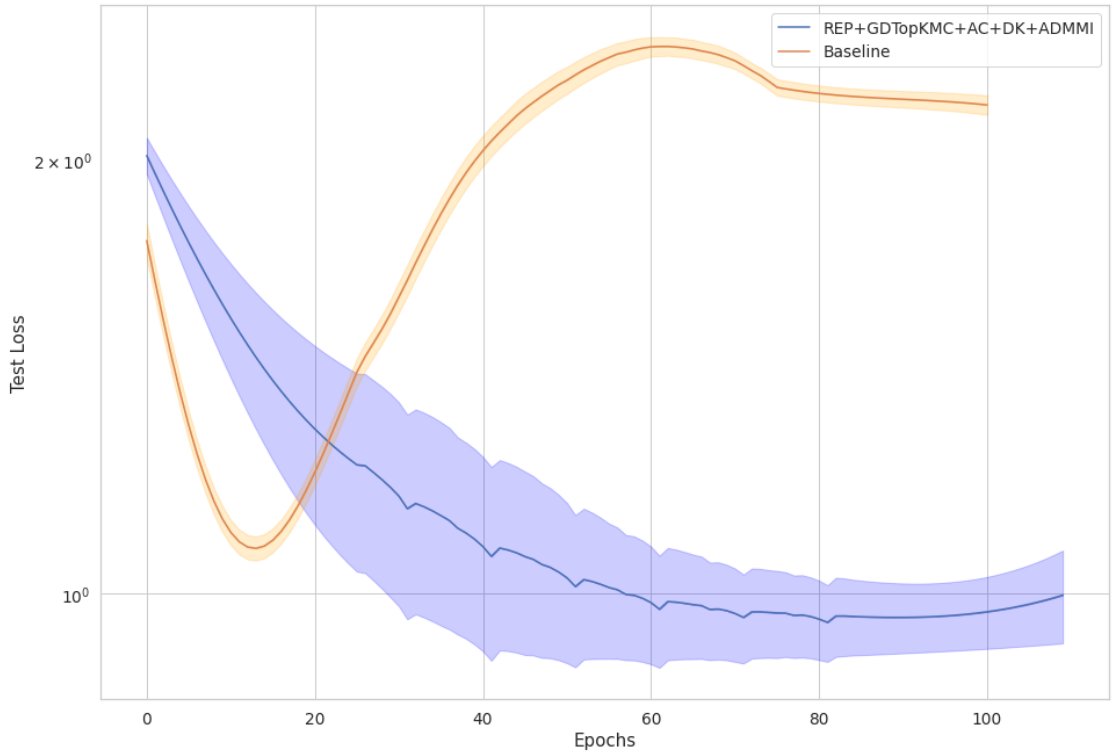


Figure 6.7.: REP+GDTopKMC+AC+DK+ADMMI test loss (explanation see Section 2.4.4) on AlexNet-s/CIFAR-10, 11 runs, mean and standard deviation, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability. The increase in test loss after reaching a minimum as observed in the baseline could be attributable to overfitting.

implications for its hardware friendliness. Regarding linear layers, Figure 6.12 shows again that REP+GDTopKMC+AC+DK+ADMMI has a distinct preference to remove adjunct weights, inducing rectangular shaped sparsity (zero) patterns, aligning non-zero values in a low number of relatively dense rows and columns.

6.3.3. Performance Improvements

6.3.3.1. Efficient LGD

For the evaluation of the memory efficient computation of LGD introduced in Section 4.1.1.2, I experimentally verified my assumptions on memory complexity by computing LGD on a set of architectures of various sizes relevant in the field as shown in Tables 3.3 using two different values for lookback lb motivated by the experiments in this thesis as well as [23, 64]. As seen in Tables 6.6 and 6.7, memory overhead is reduced by up to 2 orders of magnitude. This improvement makes it possible to use LGD (but also GD

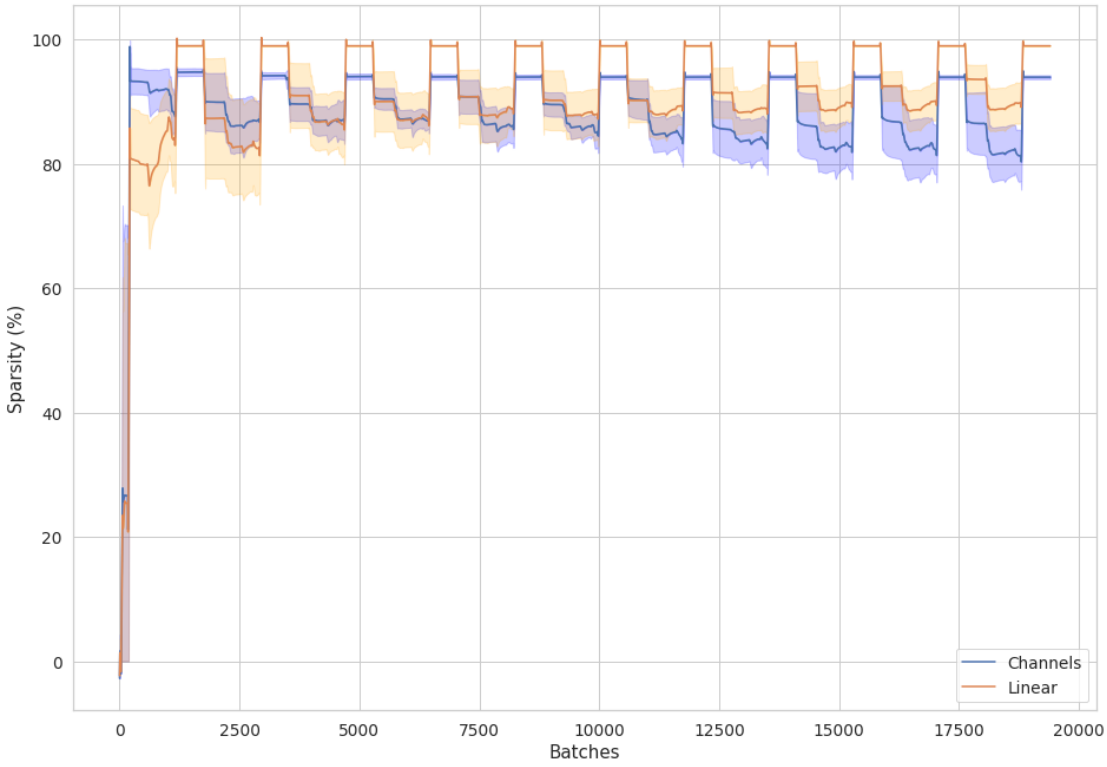


Figure 6.8.: REP+GDTopKMC+AC+DK+ADMMI development of linear and channel sparsity during training of, 11 runs, mean and standard Deviation, AlexNet-s/CIFAR-10, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability.

Model	**GD (old), lb = 45**	**GD (old), lb = 90**	**GD (proposed)**
LeNet-5	0.09	0.18	0.002
AlexNet-S	4.905	9.81	0.109
AlexNet	10.98	21.96	0.244
ResNet18	2.115	4.23	0.047
ResNet20	0.045	0.09	0.001
ResNet32	0.09	0.18	0.002
ResNet34	3.915	7.83	0.087
ResNet44	0.135	0.27	0.003
ResNet50	4.59	9.18	0.102
ResNet56	0.18	0.36	0.04

Table 6.6.: Gigabytes of memory overhead (stemming from the requirement to store gradients, see Section 4.1.1.3) of gradient diversity (GD) computation for various values of lookback *lb* on different DNN architectures.

Figure 6.9.: REP+GDTopKMC+AC+DK+ADMMI layer sparsity on ResNet32/CIFAR-10

in general) for various architectures that could not previously be examined by this metric. As can be seen in Table 6.1, even contemporary high-performance hardware could not have handled the large GPU memory overhead of GD computations with large values of lb for certain models such as VGG-8/11/13/16, AlexNet or WRN-28-10 without resorting to either continuously shuffling collected gradients between system memory and GPU for GD computation or even completely reallocating collected gradients to system memory and computing GD entirely on the CPU. With the proposed method of GD computation, however, even modest consumer-grade hardware can handle GD computation even for very large values of lb on the GPU entirely, allowing for a much more efficient use of resources.

6.3.3.2. Overhead Reduction through MC

In an evaluation of the concept introduced in Section 4.1.1.6, the algorithmic overhead of GDTopK and GDTopKMC was measured on the LeNet-5/MNIST and AlexNet-s, ResNet-18/CIFAR-10 architecture/dataset combinations. As can be seen in Table 6.8, the

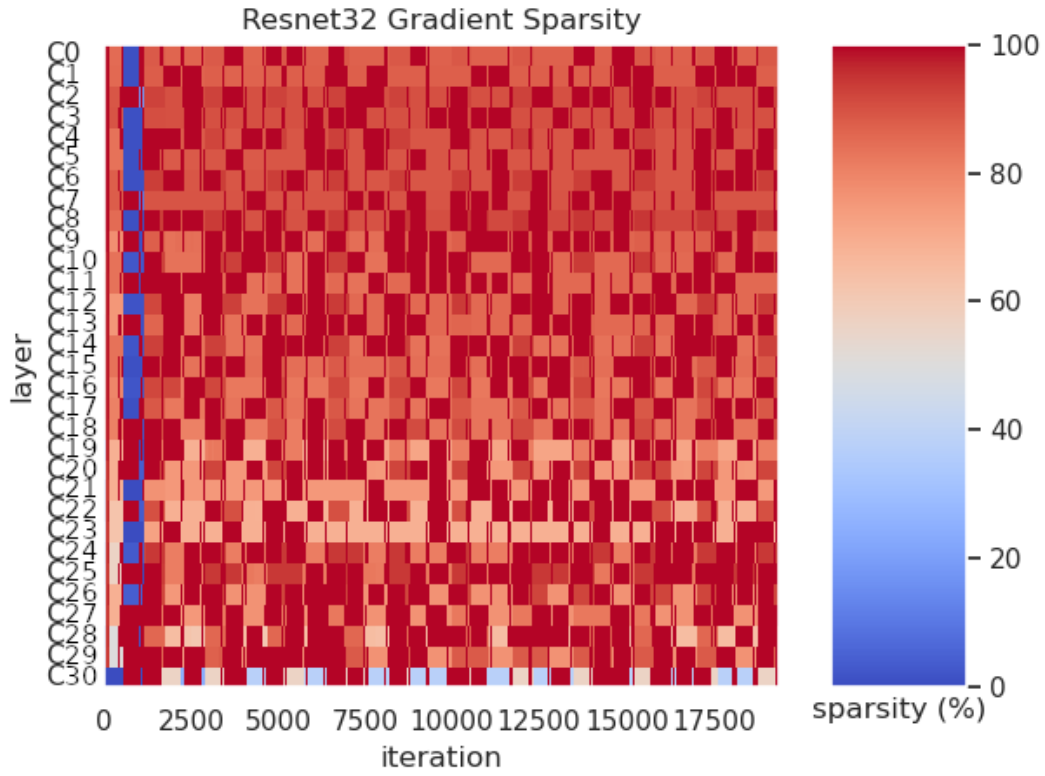


Figure 6.10.: REP+GDTopKMC+AC+DK+ADMMI gradient sparsity on ResNet32/CIFAR-10

overhead of GDTopK is effectively reduced through Monte Carlo sampling in GDTopKMC by up to 2 orders of magnitude without significant reduction of gradient sparsity or accuracy.

6.3.4. Training and Inference Acceleration

6.3.4.1. Comparison with Primary Competitors

Table 6.9 compares REP+GDTopKMC+AC+DK+ADMMIs performance to competing works listed in Table 3.6 which specifically seek to accelerate training. In terms of Δ Acc, REP+GDTopKMC+AC+DK+ADMMI outperforms competing works in 42.67% of the experiments and is itself outperformed in 42.67% of the experiments. In 14.66% of the experiments, results are inconclusive, with neither REP+GDTopKMC+AC+DK+ADMMI nor the competing works being clearly superior. With respect to channel/filter pruning rate C/F, REP+GDTopKMC+AC+DK+ADMMIs compares favourably in 8.33% of experiments, but for 91.67% of the experiments, comparable data is missing. In terms of overall pruning ratio P, REP+GDTopKMC+AC+DK+ADMMI is superior in 16.67%

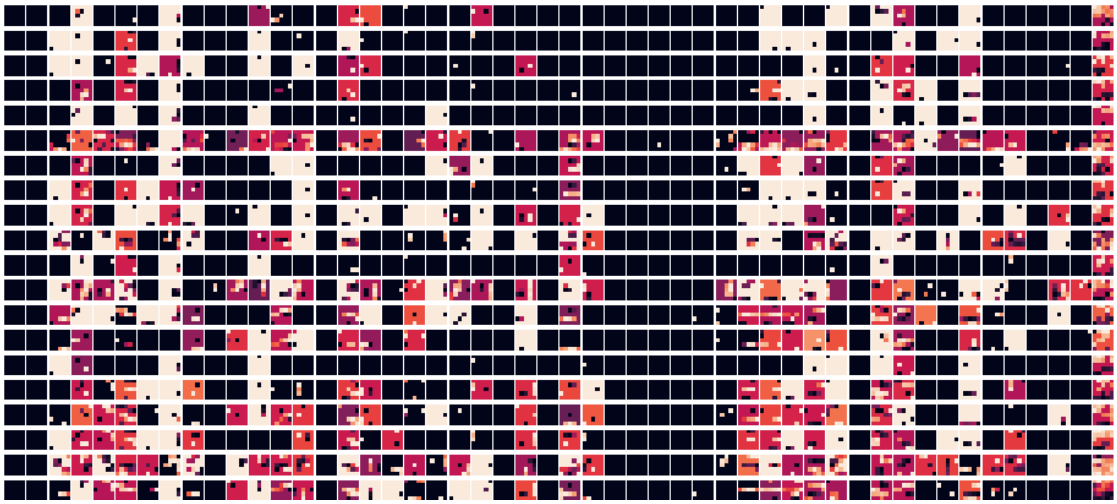


Figure 6.11.: REP+GDTopKMC+AC+DK+ADMMI induced sparsity structure in convolutional layer 2 of LeNet-5 trained on MNIST, black areas indicate zero elements/filters, colored areas indicate non-zero elements/filters.

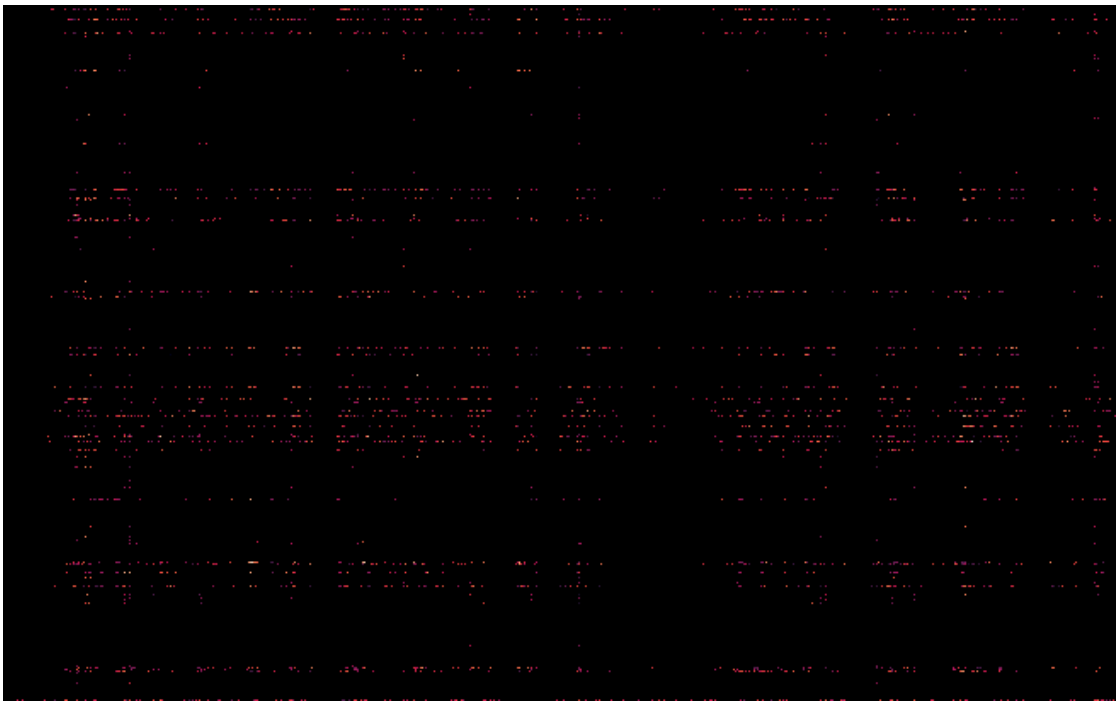


Figure 6.12.: REP+GDTopKMC+AC+DK+ADMMI induced sparsity structure in linear layer 1 of LeNet-5 trained on MNIST, black areas indicate zero elements, colored areas indicate non-zero elements.

Model	GD (old), lb = 45	GD (old), lb = 90	GD (proposed)
DenseNet121	1.44	2.88	0.032
DenseNet161	5.175	10.35	0.115
DenseNet169	2.565	5.13	0.057
DenseNet201	3.6	7.2	0.08
MobileNetV2	0.63	1.26	0.014
MobileNetV3-S	0.45	0.9	0.01
WRN-16-8	2.07	4.14	0.046
WRN-16-10	3.195	6.39	0.071
WRN-22-8	3.195	6.39	0.071
WRN-28-10	6.66	13.32	0.148
VGG-8	22.95	45.9	0.51
VGG-11	23.895	47.79	0.531
VGG-13	23.94	47.88	0.532
VGG-16	24.885	49.77	0.553

Table 6.7.: Gigabytes of memory overhead (stemming from the requirement to store gradients, see Section 4.1.1.3) of gradient diversity (GD) computation for various values of lookback *lb* on different DNN architectures.

of cases while results are inconclusive or incomparable due to missing data in 8.33% and 58.33% of cases respectively. While in terms of inference acceleration FLOPs(I), REP+GDTopKMC+AC+DK+ADMMI is clearly outperformed by competing works, in the target metric training acceleration FLOPs(T), REP+GDTopKMC+AC+DK+ADMMI outperforms competitors in 58.33% of cases and is itself outperformed in only 16.67% of cases. In 16.67% of cases, missing data does not allow for a comparison and in 8.33% of cases, results are inconclusive.

6.3.4.2. Comparison with Secondary Competitors

With regard to works listed in Table 3.5 that do not specifically target or document training acceleration, but that might have a potential positive effect, Table 6.10 shows a comparison with REP+GDTopKMC+AC+DK+ADMMI. REP+GDTopKMC+AC+DK+ADMMI compares favourably in terms of Δ Acc in 28.57% of cases, while it is outperformed by its competitors in 57.14% of cases. In terms of channel/filter pruning rate C/F, REP+GDTopKMC+AC+DK+ADMMI shows better results than its competitors in 42.86% of experiments yet in 57.14% of cases, no comparison can be made due to missing data. Concerning overall pruning ratio P, REP+GDTopKMC+AC+DK+ADMMI shows a distinctive advantage, outperforming other works in 71.43% of experiments while yielding inconclusive results in 28.57% of cases. Regarding inference acceleration FLOPs(I), 28.57% of experiments produced results in favor of REP+GDTopKMC+AC+DK+ADMMI while in the other cases (71.43%), insufficient data did not allow for a comparison.

Data	Model	Approach	Δ Acc	⊘ G(T)	⊘ Rel. Overhead
MNIST	LeNet-5	GDTopK	−0.30 p.p.	88.96%	$1.43 \cdot 10^{-3}$%
		GDTopKMC	−0.15 p.p.	76.68%	$3.00 \cdot 10^{-4}$%
CIFAR-10	AlexNet-S	GDTopK	−0.86 p.p.	90.16%	$3.70 \cdot 10^{-4}$%
		GDTopKMC	−1.49 p.p.	88.70%	$1.00 \cdot 10^{-5}$%
	ResNet-18	GDTopK	+1.39 p.p.	73.64%	$2.78 \cdot 10^{-5}$%
		GDTopKMC	+2.29 p.p.	70.36%	$8.00 \cdot 10^{-7}$%

Table 6.8.: Overhead reduction through MC sampling in GDTopK compared to GDTopKMC. ⊘ Rel. Overhead = average percentage of the total FLOPs ocurring during training that were spent on the pruning heuristic and pruning mask application, ⊘ G(T) = average gradient sparsity during training.

6.3.5. Gradient Sparsity

An interesting property of training with REP+GDTopKMC+AC+DK+ADMMI is the high amount of sparsity induced in the gradients that are used for the weights updates during training. As seen in Table 6.11, REP+GDTopKMC+AC+DK+ADMMI induces 83.45% to 98.28% average sparsity gradient during training, which could be of use in a distributed (i.e., multi-node) training setting. As described in Section 2.6.2, in distributed training local nodes upload their gradients to a global parameter server which updates a global model based on the aggregate of all gradients of the local model instances. In this context, a high degree of sparsity in the local gradients could lead to significant reductions in communication overhead; hence future research directions should include an adaption of REP+GDTopKMC+AC+DK+ADMMI to the multi node setting and comparison with State-of-the-Art works in this domain.

6.4. Assumptions and Limitations

6.4.1. Hyperparameter Sensitivity Study

During the Hyperparameter sensitivity study in Section 6.3.1.2, the combinations of hyperparameters evaluated were manually selected by the author to explore those combinations deemed most interesting, hyperparameter distributions such as shown in Table 6.5 were computed ex-post. This methodological approach obviously introduces a certain bias. A methodologically more clean approach would be to define hyperparameter ranges ex-ante and randomly draw combinations from these distributions. However, this could produce experiments that either fail or have very similar results, leading to the number of experiments required in total to increase beyond what could be executed in an acceptable timescale. Consequently the author decided to limit the scope of the hyper parameter study and transfer the task of a more exhaustive evaluation to future work.

Data	Model	Paper	Δ Acc	C/F	P	FLOPs(I)	FLOPs(T)
MNIST	LeNet-5	EP	−0.13 p.p.	?	45.95%	?	38.06%
		MY	−0.54 p.p.×	91.77%?	98.60%√	91.68%?	83.82%√
	LeNet-A	DP	−0.10 p.p.	?	81.24%[4]	?	?
			−1.17 p.p.	?	98.12%[4]	?	?
			−2.43 p.p.	?	99.44%[4]	?	?
		MY	−0.64 p.p.∼	0.00%?	48.86%×	48.86%?	79.53%?
CIFAR-10	ResNet-20	EF	−0.37 p.p.	?	64.90%	63.90%	52.15%[6]
		MY	−4.29 p.p×	72.16%?	71.94%√	46.32%×	62.36%√
	ResNet-32	EP	+0.13 p.p.	?	66.44%	?	44.86%
		PT	−1.80 p.p.	?	?	66.00%	53.00%
		MY	−4.22 p.p.×	71.20%?	71.00%?	46.03%×	63.64%√
	ResNet-50	PT	−1.10 p.p.	?	?	50.00%	70.00%
		MY	−1.08 p.p.√	82.20% ?	80.75%?	47.35%×	70.71%√
	VGG-S	DP	+0.33 p.p.	?	66.67%[4]	?	?
			+0.18 p.p.	?	80.00%[4]	?	?
			−3.41 p.p.	?	95.00%[4]	?	?
			−10.80 p.p.	?	96.67%[4]	?	?
		PR	+0.10 p.p.	?	80.67%	57.99%	?[5]
		MY	−2.55 p.p.∼	75.39%?	88.74%∼	36.49%?	57.96%?
	VGG-11	PT	−0.70 p.p.	?	?	57.00%	65.00%
		MY	−0.36 p.p.√	95.94%?	98.64%?	46.59%×	65.56%√
	VGG-13	PT	−0.60 p.p.	?	?	56.00%	63.00%
		MY	+0.42 p.p.√	97.04%√	98.46%?	46.58%×	65.02%√
	VGG-16	EF	−0.30 p.p.	?	96.60%	91.30%	88.49%[6]
		MY	+6.54 p.p.√	97.01%?	97.26%√	46.56%×	66.33%×
CIFAR-100	ResNet-32	PT	−1.40 p.p.	?	?	32.00%	46.00%
			−0.70 p.p.	?	?	53.00%	69.00%
		MY	−9.08 p.p.×	86.37%?	85.75%?	48.86%∼	64.01%∼
	VGG-11	PT	−1.30 p.p.	?	?	47.00%	57.00%
		MY	−3.03 p.p.×	77.76%?	89.86%?	30.80%×	57.44%√
	VGG-13	PT	−1.10 p.p.	?	?	42.00%	52.00%
		MY	+10.38 p.p.√	75.71%?	89.90%?	29.34%×	51.56%×

Table 6.9.: Performance comparison of works specifically targeting training and effect documented in original paper. Fields with % read as reduced by X%. [6]Unit of measurement no specified, FLOPs appears likely though within the context of the paper, LeNet-A=LeNet-300-100, VGG-S=VGG-8, VGG-C/D=VGG-16. [4]Percentage of tracked parameters instead of pruning rate. The symbols indicate that **MY** approach is √ better than all, ∼ better than some, × better than none of the other approaches evaluating on the same architecture/dataset combination in the respective column or ? can not be compared due to insufficient data. **MY** denominates REP+GDTopKMC+AC+DK+ADMMI. FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Data	Model	Paper	Δ Acc	C/F	P	FLOPs(I)	FLOPs(T)
MNIST	LeNet-5	SNIP	−0.80 p.p.	?	99.00%	?	?
		AP	−0.02 p.p.[3]	?	?	93.00%	?
			−0.02 p.p	?	99.53%	?	?
			−0.00 p.p	?	99.68%	?	?
		GSM	+0.01 p.p.	?	99.20%	?	?
			−0.15 p.p.	?	99.67%	?	?
		MY	−0.54 p.p.∼	91.77%?	98.6%∼	91.68%?	83.82%?
	LeNet-A	SNIP	−1.60 p.p.	?	98.00%	?	?
		AP	−0.22 p.p.[3]	?	?	91.00%	?
			+0.06 p.p.	?	98.75%	?	?
		GSM	±0.00 p.p.	?	98.34%	?	?
		LTH	?	?	20.00%	?	?
		MY	−0.64 p.p.∼	0.00%?	48.86%∼	48.86%?	79.53%?
CIFAR-10	AlexNet-s	SNIP	−0.87 p.p.	?	90.00%	?	?
	AlexNet-b		−0.58 p.p.	?	90.00%	?	?
	AlexNet-s	**MY**	−1.90 p.p.×	93.81%?	98.45%√	46.12%?	64.57%?
	ResNet-18	LTH	?	20.00%	19.71%	?	?
	ResNet-18	**MY**	−1.42 p.p.×	90.48%√	90.15%√	47.72%?	66.95%?
	ResNet-20	VCP	−0.35 p.p.	38.00%	20.41%	16.47%	?
		MY	−4.29 p.p×	72.16%√	71.94%√	46.32%√	62.36%?
	VGG-C	SNIP	−0.45 p.p.	?	95.00%	?	?
	VGG-D		−0.33 p.p.	?	95.00%	?	?
	VGG-16	VCP	−0.07 p.p.	62.00%	73.34%	39.10%	?
		MY	+6.54 p.p.√	97.01%?	97.26%√	46.56%?	66.33%?
CIFAR-100		VCP	+0.07 p.p.	32.00%	37.87%	18.05%	?
		MY	+10.44 p.p.√	77.63%√	87.92%√	36.47%√	56.94%?

Table 6.10.: Performance comparison of works with possible effect during training but not documented in original paper. Fields with % read as reduced by X%. [3]Neuron Pruning, LeNet-A=LeNet-300-100, VGG-C/D=VGG-16. The symbols indicate that **MY** approach is √ better than all, ∼ better than some, × better than none of the other approaches evaluating on the same architecture/dataset combination in the respective column or ? can not be compared due to insufficient data. **MY** denominates REP+GDTopKMC+AC+DK+ADMMI. FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Data	**Model**	**Δ Acc**	**⊘ G(T)**
MNIST	LeNet-5	−0.54 p.p.	87.27%
	LeNet-A	−0.64 p.p.	93.36%
CIFAR-10	AlexNet-S	−1.90 p.p.	91.78%
	ResNet-18	−1.42 p.p.	91.63%
	ResNet-20	−4.29 p.p.	86.57%
	ResNet-32	−4.22 p.p.	85.50%
	ResNet-50	−1.08 p.p.	91.50%
	VGG-8	−2.55 p.p.	97.68%
	VGG-11	−0.36 p.p.	96.23%
	VGG-13	+0.42 p.p.	96.76%
	VGG-16	+6.54 p.p.	98.28%
CIFAR-100	ResNet-20	−3.37 p.p.	83.45%
	ResNet-32	−9.08 p.p	87.99%
	ResNet-50	−2.43 p.p.	85.92%
	VGG-8	−6.49 p.p.	97.20%
	VGG-11	−3.03 p.p.	96.55%
	VGG-13	+10.38 p.p.	96.18%
	VGG-16	+10.44 p.p.	98.22%

Table 6.11.: Average gradient sparsity (⊘ G(T)) of REP+GDTopKMC+AC+DK+ADMMI training

6.4.2. Convergence Analysis

The provided convergence analysis (see Section 6.3.2) is limited to preliminary empirical evaluation of a few selected cases. For reasons similar to those leading to the decision to reduce the scope of the hyper parameter sensitivity study (see Section 6.4.1), the author decided to set a reduced scope for the convergence analysis. For a more complete picture, it would be required to thoroughly evaluate convergence behaviour, at least on all relevant datasets and architectures noted in Tables 6.9 and 6.10. Moreover, another interesting perspective on convergence would be to not train for a fixed number of epochs and measure the resulting accuracy as was done in this thesis but instead to fix accuracy to a given value (e.g. one obtained by a baseline or a competing work) and measure how many epochs of training are required to attain this accuracy. Such a perspective would provide valuable insights into convergence speed. Further, the convergence analysis provided lacks a formal analytical part (i.e., proof of convergence or boundaries for convergence rates) as well. Proofs regarding convergence however are also rare in literature (compare Section 3.4.2.11, only 10.5% of the surveyed works provided abstract guarantees at all which includes convergence guarantees). Both, an exhaustive empirical evaluation as well as an analytical exploration of the proposed algorithms convergence behaviour, should be subject to future research though.

6.4.3. Training, Hyperparameter Optimization and Model Selection

As described in Section 6.3.1, a standardized training approach of 100 epochs for large architectures trained on CIFAR-10/100 and 10 epochs for small architectures training on MNIST was chosen. As noted in Section 6.2.4, this standardized training approach was necessary due to differences in training epochs as well as batch sizes found in related work. The concrete numbers were motivated by experiences I made with training some of the architecture/model used in this thesis in previous work [23]. While successfully limiting computational requirements, particularly for large architectures and slowly converging architectures such as VGG types, this approach has the downside of not necessarily training each architecture/dataset combination to its highest attainable test accuracy value (see also Section 6.4.2). Consequently, the large positive increases in Δ Acc observed in Table 6.9 could be attributable to experimental setup instead of REP+GDTopKMC+AC+DK+ADMMI truly being capable to increase accuracy scores in the double-digit range in exhaustively trained networks.

Regarding model selection and hyperparameter optimization, no specialized approach was employed in order to save computation time and the results depicted in Tables 6.9 and 6.10 do not necessarily represent the best possible results that could be obtained by REP+GDTopKMC+AC+DK+ADMMI but rather a lower bound. Most likely, model selection approaches such as K-fold cross validation [2, p. 180] combined with randomized hyperparameter search could yield even more convincing results.

6.4.4. Performance Model

As seen in Tables 6.9 and 6.10, despite comparable pruning rates, acceleration in terms of reduction of training FLOPs does not reach levels achieved by some competing works despite comparable pruning rates. This might be a result of the simplified performance model. For example [11] uses specialized libraries to fully simulate the execution of every single operation on purpose-built sparse hardware designed and optimized to be used in conjunction with the authors proposed pruning algorithm, [227] employs a prototypical FPGA realization of hardware optimized to the utilization of their pruning approach and [278] as well as [230], while evaluating on GPUs that do not necessarily exploit sparsity optimally, had sufficient homogeneous hardware (i.e. same GPU type used for all experiments) available to allow for comparable measurements, which was not the case in the evaluation of this thesis, see Section 6.2.1. The performance model used in this thesis solely takes into account FLOPs and hence speed-ups achieved through pruning of convolutional or linear layers. Operations occurring in, for example, batch normalization and average pooling layers are neglected for simplicity and hence a reduction of computational complexity in these layers is not documented.

6.4.5. Missing Experiments

Given the time constraints of this thesis as well as the limited hardware available, only certain selected experiments could be conducted in Section 6. The missing experiments required for a full comparison to related works are listed in Tables A.17 and A.18.

7. Contribution and Delimitation

The contribution of this work, besides its exhaustive literature survey provided in 3.4, lies in exploring research questions RQ1 - RQ4 and RQ7 (see Section 3.4.2) as well as improving memory efficiency of gradient diversity in Section 4.1.1.2 in the process.

7.1. Literature Survey

In the literature survey, I first provide a brief overview of historic developments (Section 3.3) in the field of neural network pruning and highlight the most influential works of the past. In Section 3.4, I contribute an extensive listing and categorization of carefully selected SotA (2017-2021) intra-training pruning works which are subjected to thorough analysis focusing on experiments, metrics and the identification of direct and potential competitors (intra-training pruning with the goal to accelerate training) to this thesis in Section 3.4.1. This focus on specific properties of SotA works distinguishes the survey that is part of this thesis from others in the field, which rather focus on distinguishing methodical approaches. The value of this survey hence lies not only in documenting these observations, but also allowing other researchers in the future to select their experiments and metrics in such a way that their works become more comparable. Surveys [110, 108, 107, 105, 101, 106, 100] do not distinguish between intra-training pruning with the goal to accelerate inference or intra training pruning with the goal to accelerate training, and do not provide an overview over the popularity of metrics and experiments used by researchers in the field of intra training DNN pruning. [332, 7] provide an overview of common experiments but do not distinguish intra and post training DNN pruning and only consider the metrics sparsity and/or speed-up and accuracy.

7.2. Answered Research Questions

Regarding RQ1 (Section 3.4.2.1), with GDTopKMC I devise a probabilistic top-k gradients training scheme in Section 4.1.1.6 which through learning the distribution of the proposed LGD based heuristic over a certain number of sampling epochs and for the rest of the training just drawing from the distribution effectively reduces the methods overhead without losing quality in the final result compared to GDTopK's direct computation of the heuristic. GDTopK, the underlying top-k gradients scheme of GDTopKMC, differs from related top-k gradients schemes [93, 11] through its heuristic as well as its very coarse and efficient granularity: instead of removing individual elements of gradients, gradient tensors are selected for deletion as a whole. GDTopKMC is different from all existing probabilistic intra training pruning techniques not only because it is the only

existing work using gradient diversity in this field (which is true for GDTopK as well), but also because it attempts to reduce the computational overhead of directly computing an originally deterministic pruning technique through the use of statistics. Furthermore [278], which is the closest competitor to GDTopKMC due to its aim at training acceleration, directly estimates gradients instead of a heuristic and is also more fine granular because it applies pruning element-wise. Of the works using statistics for pruning that are not aiming at training acceleration but the acceleration of inference, [112] is implemented as a modified BN layer and [183] uses an extension of variational dropout; hence they are different from GDTopKMC in their architectural integration into training.

GDTopK or GDTopKMC in conjunction with the information theory motivated REP weights pruning method introduced in Section 4.1.2.4 answers RQ2 (Section 3.4.2.2) i.e., that using separate pruning methods for forward and backwards passes is advantageous in terms of training accuracy and/or speedup compared pruning methods that only affect one of the two phases of training such as those listed in Tables 3.9 and 3.10. The proposed technique is also different from works combining sparse forward and backwards passes [93, 11] because no separate heuristics for accelerating forward and backward passes are used in these works.

By combining GDTopK or GDTopKMC and REP, which are both structured pruning methods, with magnitude pruning MAG and ADMMR/ADMMI as described in Section 4.2, I show that model compression and training acceleration can be improved through the combination of different types of structured pruning and unstructured pruning, answering RQ3 (Section 3.4.2.3) in the process. This constitutes a novelty because combining structured and unstructured pruning during training has not yet been done in works related to intra-training pruning for training acceleration as discussed in Section 3.4.2.3.

REP as introduced in Section 4.1.2.4, which is based on LGD, investigates RQ4 (Section 3.4.2.4) as it provides a structured pruning heuristic for intra-training pruning which easily integrates with standard SGD and potentially similar other popular gradient-based optimization techniques without relying directly on the norm of the networks weights or gradients. This makes REP different from other intra-training pruning techniques aimed at training acceleration: [278] does not train via standard chain rule-based SGD for optimization, [215] uses a penalty term rather than heuristics while [93] and [11] both rely on the norm of accumulated gradient for determining the network's sensitivity to pruning. [227] proposes a pruning algorithm based on weights magnitude instead of norm, but this forces it to prune in an unstructured manner and [219], while using geometric median instead of norm as heuristic for filter pruning, is aimed at inference acceleration.

Through combining GDTopK or GDTopKMC and REP with ADMMR/ADMMI in Section 4.2, RQ7 (Section 3.4.2.7) is partially answered which seeks to explore the effects of combining intra-training pruning methods for training acceleration with methods specifically devised to accelerate inference through pruning. Since there is a large number of inference acceleration aimed intra-training pruning works potentially capable of accelerating training as well (see Tables 3.9 and 3.10, the evaluation of a single approach is naturally only a small part of the investigation of RQ7).

7.3. Integration into Scientific Landscape

The core methods developed in this thesis are REP (Section 4.1.2.4) and GDTopK/GDTopKMC (Section 4.1.1.5), hence the argument for the integration into the contemporary scientific landscape will revolve around these methods. REPs and GDTopK/GDTopKMCs integration into the scientific landscape described in Chapter 3 is visualized in Figures 7.1 and 7.2 at 8 different levels, whereby level 0 represents the most coarse categorization and level 7 represents the most fine granular categorization. REP is a method designed for accelerating DNN training and, as a side effect due to pruning being focused on the networks weights and filters, also carries over advantages in terms of acceleration and compression over to the inference phase. Because the inference acceleration associated with REP is not part of its initial design goal, it is coloured dashed in Figure 7.1 level 3. REP is a sensitivity heuristic method based on information theory but not directly employing it, information theory is coloured dashed in Figure 7.1 level 4. REP is applicable to convolutional and linear layers weights, where it induces filter level as well as (block-) structured sparsity combined with element wise sparsity through additional magnitude pruning. Although gradients are as well pruned by REP to accelerate the expensive backwards pass, the mechanism serves mainly to limit the pruned weights growing back after the aggressive pruning phase in the beginning, i.e., REPs pruning of gradients is more a necessary design constraint rather than an initial design goal, which is why it is coloured dashed in Figure 7.1.

GDTopK/GDTopKMC are both methods focused on accelerating training through gradient pruning while leaving the weights of the network untouched. GDTopK/GDTopKMC employ a sensitivity-based pruning heuristic to convolutional and linear layers gradients, which are pruned on a per-layer basis; hence the approach is a very coarsely structured type. Because the weights of the network are not pruned by neither GDTopK nor GDTopKMC, it does not carry over any advantage to the training phase.

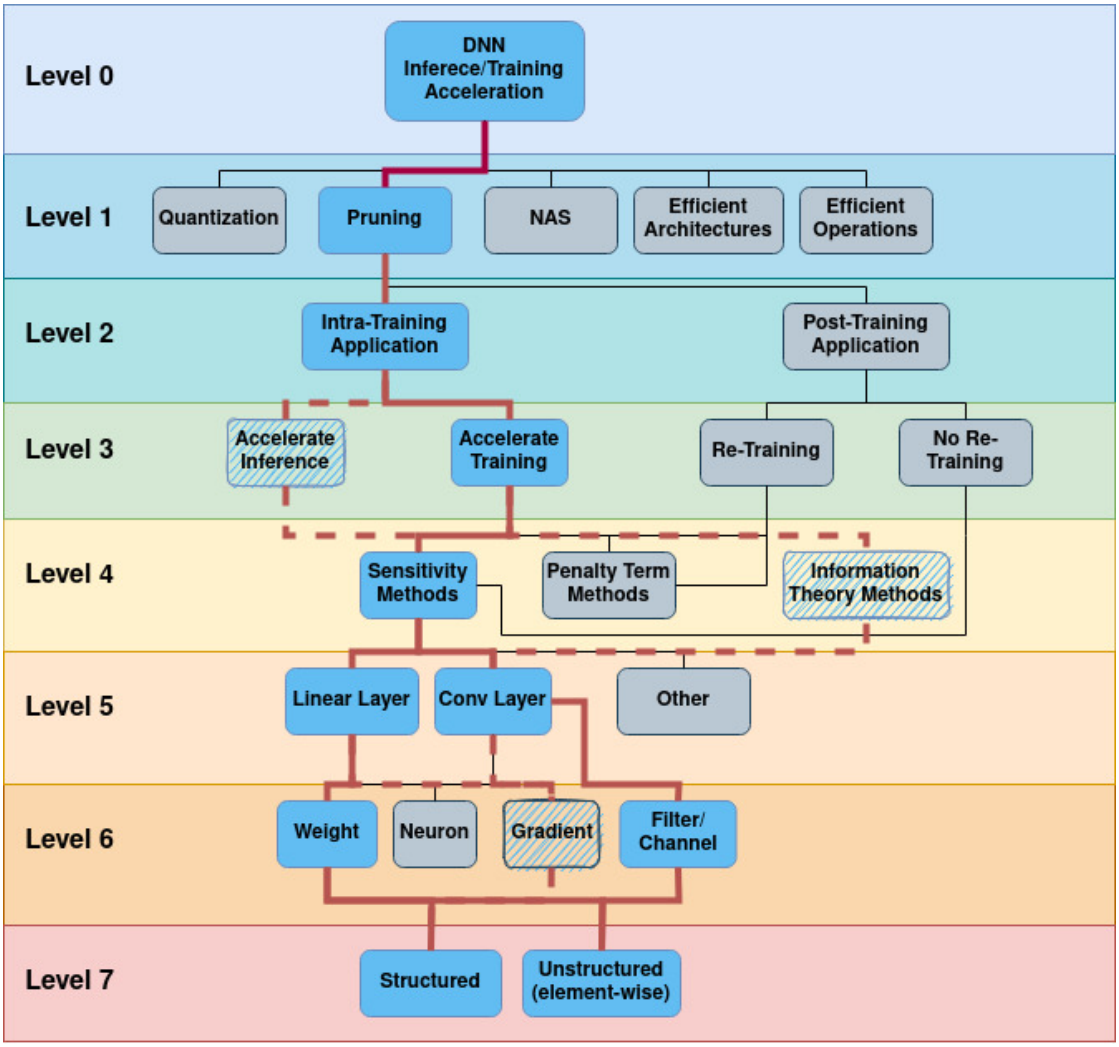


Figure 7.1.: REP integration into scientific landscape. REP falls into the categories coloured in blue. Categories dashed in blue partially fit REP. Grey categories are not directly related to REP. A bold, red line indicates the relation between two categories which are related to REP. If the line is dashed, it represents a connection to a category in which REP only partially fits into.

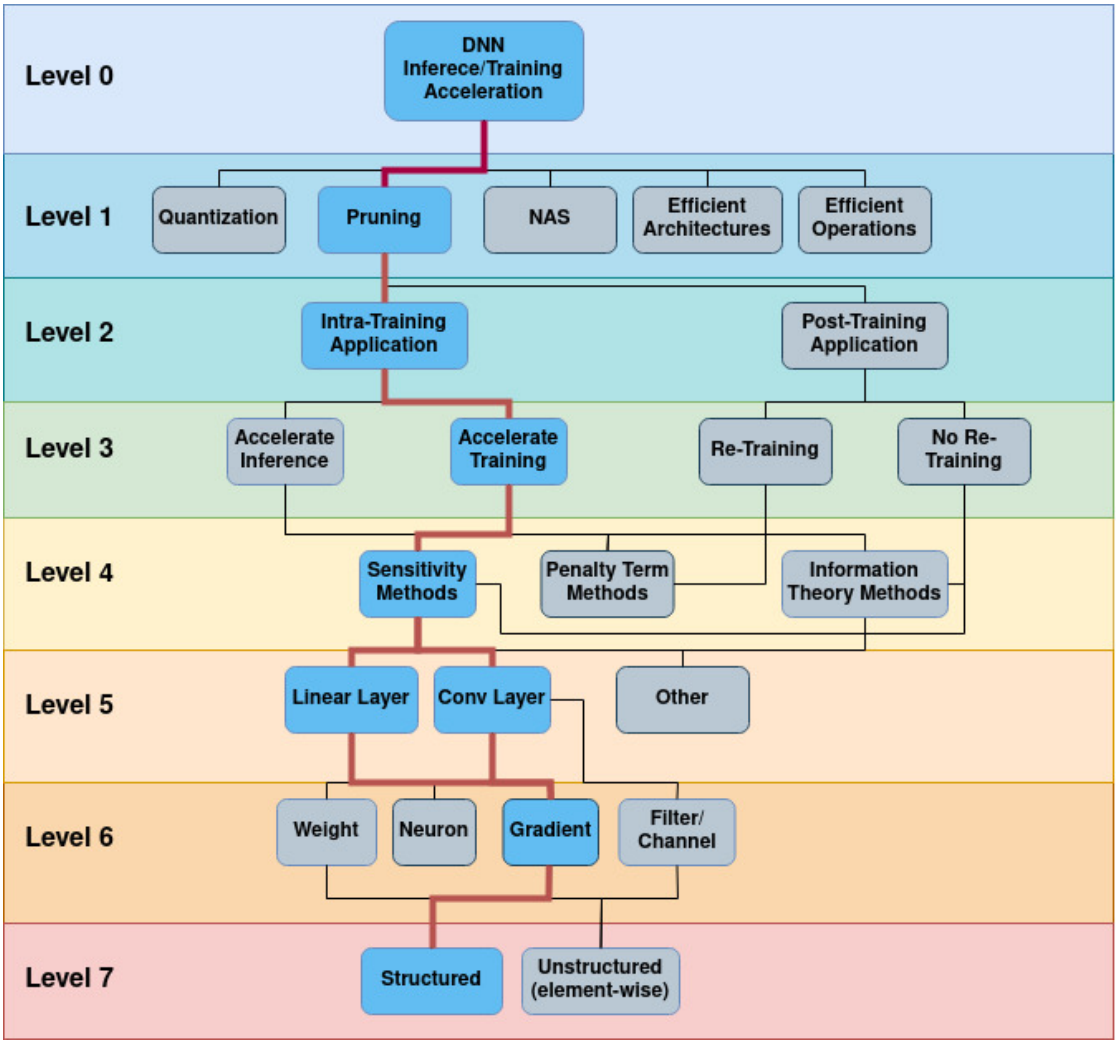


Figure 7.2.: GDTopK and GDTopKMC integration into scientific landscape. GDTopK and GDTopKMC fall into the categories coloured in blue. Grey categories are not directly related to GDTopK and GDTopKMC. A bold, red line indicates the relation between two categories which are related to GDTopK and GDTopKMC.

8. Conclusion

This thesis presents an exhaustive documentation of the State-of-the-Art in intra-training DNN pruning (Section 3.4). In the beginning, a brief overview of training acceleration approaches alternative to pruning as well as a brief historical contextualization is provided. For the SotA, the surveyed works were selected by relevance or citation count dependent on availability in the results provided by multiple academic search engines (BASE, CORE, Semantic Scholar, Microsoft Academic, Google Scholar, IEEE Xplore, ACM Digital Library, NeurIPS). Key insights in the literature include the finding that the majority of works applying intra-training pruning focus on accelerating inference (97.78%), that only a small number (14.33%) prune during training with the intent of accelerating training and that a certain number of works (28.88%) targeting inference could also be conceptually capable of accelerating training, but authors do not document any training advantages, see Section 3.4.1.4. The majority of works further employ empirical methodology to demonstrate their contributions advantages, with only 10.5% of the surveyed works including any sort of formal proof. Regarding the experiments and metrics used in the empirical work common in the field, the survey shows that there is a clear tendency towards certain architecture/dataset combinations as well as metrics, but no standardized setup exists, hampering comparability between different approaches, see Section 3.4.1.

Based on these papers, a series of relevant research questions (Section 3.4.2) is derived. A certain subset of key research questions which is described in detail in Section 4 is answered by a novel and innovative multi directional intra-training pruning approach to accelerate DNN training. The answered research questions address the reduction of overhead during intra-training pruning, the use of different heuristics tailored specifically to the forwards- and backwards-pass of DNN training, the combination of unstructured and structured pruning for training acceleration as well as the adaption of an intra training pruning scheme originally targeted at inference acceleration for intra training use.

The proposed approach, which combines structured and unstructured pruning, is designed to heuristically address the specific requirements of pruning DNNs weights for forward pass acceleration without, in an analogy to the initially presented biological motivation, "forgetting" already learnt concepts as well as the specific requirements of pruning gradients for accelerating computationally expensive backwards passes without losing the ability to learn, see Sections 4.1.1 and 4.1.2. Furthermore, certain novel concepts are introduced to ensure of the efficiency of the applied heuristics, reducing computational complexity of certain components by up to 2 orders of magnitude and memory complexity by up to 3 orders of magnitude, see Section 6.3.3.1 and 6.3.3.2. Through extensive empirical evaluation in Section 6.3.4, it is demonstrated that the proposed approach reaches or surpasses State-of-the-Art levels in accuracy degradation in at least 57.33% of the evaluated cases whereby in 46.67% of cases clear superiority is demonstrated and

in 14.66% the proposed approach is at least not clearly inferior. Concerning training acceleration measured in the percentage of FLOPs saved compared to a baseline, the proposed approach reaches or surpasses State-of-the-Art training acceleration in 75.00% of cases, whereby in 58.33% of cases clear superiority is demonstrated and in 16.67% the proposed approach is at least not clearly inferior. Additionally, an ablation study is provided in Section 6.3.1. In the ablation study the contribution of each of the individual components of the multi-directional pruning approach is evaluated. It shows that the high degrees of acceleration and accuracy displayed in the final results can only be reached by the combination of different algorithms treating specific problems of intra-training pruning separately.

Finally, the contribution, delimitation and integration into the scientific landscape of this work is described in Section 7. Regarding delimitation, it is argumentatively shown that no other work among those found in the literature survey employs the unique methodological approach to intra training DNN pruning proposed in this thesis. The main contribution of this thesis lies in the discovery and exploration of the above-mentioned subset of research questions through a new pruning approach which incorporates several novel techniques. Additionally, the observations regarding experiments made in the survey of SotA works could enable other researchers in the future to select their experiments and metrics in such a way that their works become more comparable such that the contribution of this thesis to the advancement of the field of intra-training DNN pruning goes beyond the mere novelty of the discovered new algorithmic approaches.

9. Future Work and Outlook

Since the proposed approach in this thesis only addresses a certain subset of the research questions outlined in Section 3.4.3, clearly the exploration of the remaining research questions listed in Section 3.4.2 is a viable approach to future work. Additionally, the provided survey suffers from some works being not completely comparable due to differences in the setup of their empirical evaluation, see Section 3.4.1.1. This makes the completion of the survey through testing these approaches under entirely comparable circumstances an attractive contribution. Furthermore, as outlined in Section 6.4.5, the empirical evaluation of this thesis still lacks completeness and hence future work should seek to complete the missing experiments listed in Tables A.17 and A.18. Additionally, certain improvements to the ablation and hyperparameter study documented in Section 6.4.3 would be possible approaches to future work. The high degree of gradient sparsity documented in Section 6.3.5 could render the multi-directional approach proposed in this thesis potentially useful in distributed training scenarios, and its adaption to such scenarios could as well be a possible direction of future research. Another possible direction could be evaluation on hardware truly capable of leveraging sparsity such as mentioned in Sections 2.6.5 and 3.4.1.3, removing the reliance on the simplifying performance model described in 5 and leading to more actionable results.

Bibliography

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, pp. 1–62, 2019.

[2] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer, 2018.

[3] R. K. Vogl, *Deep Learning Methods for Drum Transcription and Drum Pattern Generation/submitted by Richard Vogl*. PhD thesis, Universität Linz, 2018.

[4] R. Karim, "Illustrated: 10 cnn architectures," 2019. Online, accessed 15.07.2020.

[5] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[6] C. Medeiros and G. A. Barreto, "A novel weight pruning method for mlp classifiers based on the maxcore principle," *Neural Computing and Applications*, vol. 22, no. 1, pp. 71–84, 2013.

[7] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[8] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, 2016.

[9] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 821–834, 2019.

[10] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks.," *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1–124, 2021.

[11] D. Yang, A. Ghasemazar, X. Ren, M. Golub, G. Lemieux, and M. Lis, "Procrustes: a dataflow and accelerator for sparse deep neural network training," in *2020*

53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 711–724, IEEE, 2020.

[12] N. Corporation, "NVIDIA DGX-1nvidia dgx-1 with tesla v100 system architecture," 2017. Online, accessed 15.06.2022.

[13] G. H. Alex Krizhevsky, Vinod Nair, "The cifar-10 dataset," 2019. Online, accessed 15.07.2020.

[14] "Pseudo code for pytorch implementation of sgd." `https://pytorch.org/docs/stable/generated/torch.optim.SGD.html#torch.optim.SGD`. Accessed: 2022-03-14.

[15] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013.

[16] "Pseudo code for pytorch implementation of adam." `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`. Accessed: 2022-03-14.

[17] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[18] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Computer Vision – ECCV 2018* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), (Cham), pp. 191–207, Springer International Publishing, 2018.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[21] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 242–252, PMLR, 09–15 Jun 2019.

[22] M. Li, M. Soltanolkotabi, and S. Oymak, "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (S. Chiappa and R. Calandra, eds.), vol. 108 of *Proceedings of Machine Learning Research*, pp. 4313–4324, PMLR, 26–28 Aug 2020.

[23] L. Kummer, K. Sidak, T. Reichmann, and W. Gansterer, "Adaptive precision training (adapt): A dynamic quantized training approach for dnns," *arXiv preprint arXiv:2012.12775*, 2021.

[24] L. Kummer, "Dynamic neural network architectural and topological adaptation and related methods–a survey," *arXiv preprint arXiv:2108.10066*, 2021.

[25] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett, "Gradient diversity: a key ingredient for scalable distributed learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 1998–2007, 2018.

[26] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[28] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[29] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[30] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, and W. E, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 21285–21296, Curran Associates, Inc., 2020.

[31] H. Noh, T. You, J. Mun, and B. Han, "Regularizing deep neural networks by noise: Its interpretation and optimization," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[33] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges, "Lerec: A nn/hmm hybrid for on-line handwriting recognition," *Neural computation*, vol. 7, no. 6, pp. 1289–1303, 1995.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[37] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 4278–4284, AAAI Press, 2017.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[42] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[44] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[45] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," 2016.

[46] A. Rajagopal, D. A. Vink, S. I. Venieris, and C.-S. Bouganis, "Multi-precision policy enforced training (muppet): A precision-switching strategy for quantised fixed-point training of cnns," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 7943–7952, PMLR, 2020.

[47] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[48] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

[49] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[50] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," *Ph. D. thesis*, 2014.

[51] C.-F. Wang, "A basic introduction to separable convolutions," 2018. Online, accessed 16.07.2020.

[52] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

[53] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1211–1220, 2019.

[54] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14095–14103, 2020.

[55] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[57] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[58] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[59] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[60] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[61] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[62] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[63] D. Lee, D. Wang, Y. Yang, L. Deng, G. Zhao, and G. Li, "Qttnet: Quantized tensor train neural networks for 3d object and video recognition," *Neural Networks*, vol. 141, pp. 420–432, 2021.

[64] A. Rajagopal, D. A. Vink, S. I. Venieris, and C.-S. Bouganis, "Multi-precision policy enforced training (muppet): A precision-switching strategy for quantised fixed-point training of cnns," *arXiv preprint arXiv:2006.09049*, 2020.

[65] M. Hopkins, M. Mikaitis, D. R. Lester, and S. Furber, "Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190052, 2020.

[66] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu, "Deep learning towards mobile applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1385–1393, IEEE, 2018.

[67] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. Online, accessed 16.07.2020.

[68] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, H. Li, T. Drummond, and Z. Ge, "Hierarchical neural architecture search for deep stereo matching," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22158–22169, 2020.

[69] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *ArXiv*, vol. abs/1611.02167, 2017.

[70] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[71] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, pp. 2902–2911, PMLR, 2017.

[72] P. Ren, Y. Xiao, X. Chang, P.-y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surv.*, vol. 54, may 2021.

[73] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2021.

[74] D. Hernandez and T. B. Brown, "Measuring the algorithmic efficiency of neural networks," *arXiv preprint arXiv:2005.04305*, 2020.

[75] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[76] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[77] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-structured dnn weight pruning–is it beneficial in any platform?," *IEEE transactions on neural networks and learning systems*, vol. PP, 2021.

[78] X. Ma, F. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "PCONV: the missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 5117–5124, AAAI Press, 2020.

[79] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 15–28, IEEE, 2018.

[80] Y. Ji, L. Liang, L. Deng, Y. Zhang, Y. Zhang, and Y. Xie, "Tetris: Tile-matching the tremendous irregular sparsity," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[81] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Proceedings*

of the 44th Annual International Symposium on Computer Architecture, ISCA '17, (New York, NY, USA), p. 548–560, Association for Computing Machinery, 2017.

[82] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[83] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 365–382, September 2018.

[84] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for mobilenets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 14–18, IEEE, 2018.

[85] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1709–1720, 2017.

[86] D. M. Loroch, F.-J. Pfreundt, N. Wehn, and J. Keuper, "Tensorquant: A simulation toolbox for deep neural network quantization," in *Proceedings of the Machine Learning on HPC Environments*, pp. 1–8, 2017.

[87] T. Zhang, Z. Lin, G. Yang, and C. De Sa, "Qpytorch: A low-precision arithmetic simulation framework." EMC2: Workshop on Energy Efficient ML and Cognitive Computing, at NeurIPS, 2019.

[88] J. Achterhold, J. M. Köhler, A. Schmeink, and T. Genewein, "Variational network quantization," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[89] S. P. Singh and D. Alistarh, "Woodfisher: Efficient second-order approximation for neural network compression," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, (Red Hook, NY, USA), Curran Associates Inc., 2020.

[90] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[91] J. Fang, Y. Sun, Q. Zhang, K. Peng, Y. Li, W. Liu, and X. Wang, "Fna++: Fast network adaptation via parameter remapping and architecture search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[92] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[93] M. Lis, M. Golub, and G. Lemieux, "Full deep neural network training on a pruned weight budget," in *Proceedings of Machine Learning and Systems* (A. Talwalkar, V. Smith, and M. Zaharia, eds.), vol. 1, pp. 252–263, 2019.

[94] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

[95] D. Sinha and M. El-Sharkawy, "Thin mobilenet: An enhanced mobilenet architecture," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, pp. 0280–0285, 2019.

[96] H.-Y. Chen and C.-Y. Su, "An enhanced hybrid mobilenet," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*, pp. 308–312, 2018.

[97] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

[98] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (A. Singh and J. Zhu, eds.), vol. 54 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 1273–1282, PMLR, 4 2017.

[99] Y. Idelbayev and M. A. Carreira-Perpinan, "An empirical comparison of quantization, pruning and low-rank neural network compression using the lc toolkit," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021.

[100] R. Reed, "Pruning algorithms-a survey," *IEEE transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[101] G. Thimm and E. Fiesler, "Pruning of neural networks," tech. rep., IDIAP, 1997.

[102] C. Neocleous and C. Schizas, "Artificial neural network learning: A comparative review," in *Hellenic Conference on Artificial Intelligence*, p. 303, Springer, 2002.

[103] L. Franco, D. A. Elizondo, and J. M. Jerez, *Constructive Neural Networks*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[104] E. Fiesler and R. Beale, *Handbook of neural computation*. CRC Press, 2020.

[105] T. Kavzoglu and P. M. Mather, "Assessing artificial neural network pruning algorithms," in *Proceedings of the 24th Annual Conference and Exhibition of the Remote Sensing Society*, pp. 9–11, 1998.

[106] I. V. Tetko, A. E. Villa, and D. J. Livingstone, "Neural network studies. 2. variable selection," *Journal of chemical information and computer sciences*, vol. 36, no. 4, pp. 794–803, 1996.

[107] M. Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networks — a comparative study," *Open Computer Science*, vol. 3, no. 3, pp. 105–115, 2013.

[108] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[109] J. Yang, A. Bouzerdoum, and S. L. Phung, "A neural network pruning approach based on compressive sampling," in *2009 International Joint Conference on Neural Networks*, pp. 3428–3435, IEEE, 2009.

[110] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5113–5155, 2020.

[111] Q. Li, C. Li, and H. Chen, *Filter Pruning via Probabilistic Model-Based Optimization for Accelerating Deep Convolutional Neural Networks*, p. 653–661. New York, NY, USA: Association for Computing Machinery, 2021.

[112] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2775–2784, 2019.

[113] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58324–58337, 2018.

[114] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 1, Morgan-Kaufmann, 1989.

[115] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990.

[116] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[117] M. Hagiwara, "Removal of hidden units and weights for back propagation networks," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, vol. 1, pp. 351–354 vol.1, 1993.

[118] K. L. Priddy, S. K. Rogers, D. W. Ruck, G. L. Tarr, and M. Kabrisky, "Bayesian selection of important features for feedforward neural networks," *Neurocomputing*, vol. 5, no. 2-3, pp. 91–103, 1993.

[119] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon.* Morgan Kaufmann, 1993.

[120] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, 1993.

[121] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon: Extensions and performance comparisons," in *Advances in Neural Information Processing Systems* (J. Cowan, G. Tesauro, and J. Alspector, eds.), vol. 6, Morgan-Kaufmann, 1993.

[122] W. Finnoff, F. Hergert, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks*, vol. 6, no. 6, pp. 771–783, 1993.

[123] L. Prechelt, "Adaptive parameter pruning in neural networks," *International Computer Science Institute-Publications-TR*, 1995.

[124] G. Thimm and E. Fiesler, "Neural network pruning and pruning parameters," in *1st Online Workshop on Soft Computing*, Citeseer, 1996.

[125] A. Engelbrecht and I. Cloete, "A sensitivity analysis algorithm for pruning feedforward neural networks," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 2, pp. 1274–1278 vol.2, 1996.

[126] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[127] J. Sum, C.-S. Leung, G. Young, and W.-K. Kan, "On the kalman filtering method in neural network training and pruning," *IEEE Transactions on Neural Networks*, vol. 10, no. 1, pp. 161–166, 1999.

[128] R. Setiono and A. Gaweda, "Neural network pruning for function approximation," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6, pp. 443–448 vol.6, 2000.

[129] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

[130] H. Honggui and Q. Junfei, "A novel pruning algorithm for self-organizing neural network," in *2009 International Joint Conference on Neural Networks*, pp. 1245–1250, 2009.

[131] Z. Zhang and J. Qiao, "A node pruning algorithm for feedforward neural network based on neural complexity," in *2010 International Conference on Intelligent Control and Information Processing*, pp. 406–410, 2010.

[132] M. G. Augasta and T. Kathirvalavakumar, "A novel pruning algorithm for optimizing feedforward neural network of classification problems," *Neural processing letters*, vol. 34, no. 3, p. 241, 2011.

[133] S. Urolagin, P. K.V., and N. V. S. Reddy, "Generalization capability of artificial neural network incorporated with pruning method," in *Advanced Computing, Networking and Security* (P. S. Thilagam, A. R. Pais, K. Chandrasekaran, and N. Balakrishnan, eds.), (Berlin, Heidelberg), pp. 171–178, Springer Berlin Heidelberg, 2012.

[134] J. Yang, J. Ma, M. J. Berryman, and P. Perez, "A structure optimization algorithm of neural networks for large-scale data sets," *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 956–961, 2014.

[135] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, vol. 28, pp. 1135–1143, 2015.

[136] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[137] J. Yang and J. Ma, "A structure optimization framework for feed-forward neural networks using sparse representation," *Knowledge-Based Systems*, vol. 109, pp. 61–70, 2016.

[138] Y. Shin, S. Lee, M. Ahn, H. Cho, S. C. Jun, and H.-N. Lee, "Noise robustness analysis of sparse representation based classification method for non-stationary eeg signal classification," *Biomedical Signal Processing and Control*, vol. 21, pp. 8–18, 2015.

[139] M. W. Spratling, "Classification using sparse representations: a biologically plausible approach," *Biological cybernetics*, vol. 108, no. 1, pp. 61–73, 2014.

[140] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units.," in *NIPS*, vol. 1, pp. 519–526, 1988.

[141] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Back-propagation, weight-elimination and time series prediction," in *Connectionist models*, pp. 105–116, Elsevier, 1991.

[142] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in neural information processing systems*, pp. 875–882, 1991.

[143] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination applied to currency exchange rate prediction," in *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp. 2374–2379, IEEE, 1991.

[144] C. Ji, R. R. Snapp, and D. Psaltis, "Generalizing smoothness constraints from discrete samples," *Neural Computation*, vol. 2, no. 2, pp. 188–197, 1990.

[145] S. Erdogan, G.-S. Ng, and P.-H. Chan, "Measurement criteria for neural network pruning," in *Proceedings of Digital Processing Applications (TENCON '96)*, vol. 1, pp. 83–89 vol.1, 1996.

[146] G. S. Ng, A. Wahab, and D. Shi, "Entropy learning and relevance criteria for neural network pruning," *International journal of neural systems*, vol. 13, no. 05, pp. 291–305, 2003.

[147] T. Huynh and R. Setiono, "Effective neural network pruning using cross-validation," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 972–977 vol. 2, 2005.

[148] S. P. Adam, G. D. Magoulas, and M. N. Vrahatis, "Direct zero-norm minimization for neural network pruning and training," in *Engineering Applications of Neural Networks* (C. Jayne, S. Yue, and L. Iliadis, eds.), (Berlin, Heidelberg), pp. 295–304, Springer Berlin Heidelberg, 2012.

[149] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[150] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[151] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning by back propagation.," 1986.

[152] M. Ishikawa, "A structural learning algorithm with forgetting of link weights," in *International 1989 Joint Conference on Neural Networks*, pp. 626–vol, IEEE, 1989.

[153] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992.

[154] J. Sietsma, "Neural net pruning-why and how," in *Proceedings of International Conference on Neural Networks, San Diego, CA, 1988*, vol. 1, pp. 325–333, 1988.

[155] J. Sietsma and R. J. Dow, "Creating artificial neural networks that generalize," *Neural networks*, vol. 4, no. 1, pp. 67–79, 1991.

[156] J. K. Kruschke, "Creating local and distributed bottlenecks in hidden layers of back-propagation networks," *Proceedings 1998 Connectionist Models Summer School*, pp. 120–126, 1988.

[157] J. K. Kruschke, "Improving generalization in backpropagation networks with distributed bottleneck," in *Proc. Int. Joint Conf. on Neural Networks*, pp. 443–447, 1989.

[158] D. Whitley, "The evolution of connectivity: Pruning neural networks using genetic algorithm," in *Proceedings of IJCNN-90*, pp. 134–137, 1990.

[159] P. Angeline, G. Saunders, and J. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[160] I. Ciuca, J. Ware, and A. Cristea, "Removing irrelevant features in neural network classification using evolutionary computations," in *Medical Informatics Europe'97*, pp. 391–395, IOS Press, 1997.

[161] S. W. Stepniewski and A. J. Keane, "Pruning backpropagation neural networks using modern stochastic optimisation techniques," *Neural Computing & Applications*, vol. 5, no. 2, pp. 76–98, 1997.

[162] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.

[163] N. T. Siebel, J. Botel, and G. Sommer, "Efficient neural network pruning during neuro-evolution," in *2009 International Joint Conference on Neural Networks*, pp. 2920–2927, 2009.

[164] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[165] J. Tu, Y. Zhan, and F. Han, "A neural network pruning method optimized with pso algorithm," in *2010 Second International Conference on Computer Modeling and Simulation*, vol. 3, pp. 257–259, 2010.

[166] S.-H. Yang and Y.-P. Chen, "An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications," *Neurocomputing*, vol. 86, pp. 140–149, 2012.

[167] H. Amin, K. Curtis, and B. H. Gill, "Dynamically pruning output weights in an expanding multilayer perceptron neural network," in *Proceedings of 13th International Conference on Digital Signal Processing*, vol. 2, pp. 991–994, IEEE, 1997.

[168] C. A. Doukim, J. A. Dargham, and A. Chekima, "Finding the number of hidden neurons for an mlp neural network using coarse to fine search technique," in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, pp. 606–609, 2010.

[169] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ICCAD '17, p. 751–756, IEEE Press, 2017.

[170] L. Meier, S. Van De Geer, and P. Bühlmann, "The group lasso for logistic regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 1, pp. 53–71, 2008.

[171] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), p. 675–678, Association for Computing Machinery, 2014.

[172] A. J. Holtmaat, J. T. Trachtenberg, L. Wilbrecht, G. M. Shepherd, X. Zhang, G. W. Knott, and K. Svoboda, "Transient and persistent dendritic spines in the neocortex in vivo," *Neuron*, vol. 45, no. 2, pp. 279–291, 2005.

[173] D. D. Stettler, H. Yamahachi, W. Li, W. Denk, and C. D. Gilbert, "Axons and synaptic boutons are highly dynamic in adult visual cortex," *Neuron*, vol. 49, no. 6, pp. 877–887, 2006.

[174] A. Attardo, J. E. Fitzgerald, and M. J. Schnitzer, "Impermanence of dendritic spines in live adult ca1 hippocampus," *Nature*, vol. 523, no. 7562, pp. 592–596, 2015.

[175] A. R. Chambers and S. Rumpel, "A stable brain from unstable components: Emerging concepts and implications for neural computation," *Neuroscience*, vol. 357, pp. 172–184, 2017.

[176] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," in *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[177] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, nov 1997.

[178] J. S. Garofolo, "Timit acoustic phonetic continuous speech corpus," *Linguistic Data Consortium, 1993*, 1993.

[179] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, pp. 2575–2583, 2015.

[180] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[181] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[182] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, p. 107–115, feb 2021.

[183] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[184] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1998.

[185] D. Donoho and X. Huo, "Uncertainty principles and ideal atomic decomposition," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2845–2862, 2001.

[186] C. Miao and H. Yu, "Alternating iteration for $l\_\{p\}(0 < p \leq 1)$ regularized ct reconstruction," *IEEE Access*, vol. 4, pp. 4355–4363, 2016.

[187] F. Li, J. M. Zurada, Y. Liu, and W. Wu, "Input layer regularization of multilayer feedforward neural networks," *IEEE Access*, vol. 5, pp. 10979–10985, 2017.

[188] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[189] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomput.*, vol. 241, p. 81–89, jun 2017.

[190] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang, "Building fast and compact convolutional neural networks for offline handwritten chinese character recognition," *Pattern Recogn.*, vol. 72, p. 72–81, dec 2017.

[191] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang, "Casia online and offline chinese handwriting databases," *2011 International Conference on Document Analysis and Recognition*, pp. 37–41, 2011.

[192] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," in *2016 American Control Conference (ACC)*, pp. 5619–5625, 2016.

[193] X. Gao, Y. Zhao, Łukasz Dudziak, R. Mullins, and C. zhong Xu, "Dynamic channel pruning: Feature boosting and suppression," in *International Conference on Learning Representations*, 2019.

[194] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *International Conference on Learning Representations*, 2018.

[195] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[196] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm with application to wavelet-based image deblurring," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 693–696, 2009.

[197] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[198] T. Liu, J. Sun, N.-N. Zheng, X. Tang, and H.-Y. Shum, "Learning to detect a salient object," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

[199] C. Yang, L. Zhang, H. Lu, X. Ruan, and M.-H. Yang, "Saliency detection via graph-based manifold ranking," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3166–3173, 2013.

[200] X. Shen, A. Hertzmann, J. Jia, S. Paris, B. L. Price, E. Shechtman, and I. Sachs, "Automatic portrait segmentation for image stylization.," *Comput. Graph. Forum*, vol. 35, no. 2, pp. 93–102, 2016.

[201] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[202] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-Shot Network Pruning Based on Connection Sensitivity," in *International Conference on Learning Representations*, 2019.

[203] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)* (E. R. H. Richard C. Wilson and W. A. P. Smith, eds.), pp. 87.1–87.12, BMVA Press, September 2016.

[204] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *ArXiv*, vol. abs/1409.2329, 2014.

[205] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha,*

Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL (A. Moschitti, B. Pang, and W. Daelemans, eds.), pp. 1724–1734, ACL, 2014.

[206] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019.

[207] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *International Conference on Machine Learning*, pp. 6682–6691, PMLR, 2020.

[208] Z. Zhou, W. Zhou, R. Hong, and H. Li, "Online filter weakening and pruning for efficient convnets," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2018.

[209] X. Liu and Z. Zeng, "Memristor crossbar architectures for implementing deep neural networks," *Complex and Intelligent Systems*, pp. 1–16, 2021.

[210] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[211] C. LIN, Z. Zhong, W. Wei, and J. Yan, "Synaptic strength for convolutional neural network," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[212] P. Vanderhaeghen and H.-J. Cheng, "Guidance molecules in axon pruning and cell death," *Cold Spring Harbor perspectives in biology*, vol. 2, no. 6, p. a001859, 2010.

[213] F. I. Craik and E. Bialystok, "Cognition through the lifespan: mechanisms of change," *Trends in cognitive sciences*, vol. 10, no. 3, pp. 131–138, 2006.

[214] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, p. 2234–2240, AAAI Press, 2018.

[215] S. Lym, E. Choukse, S. Zangeneh, W. Wen, S. Sanghavi, and M. Erez, "Prunetrain: Fast neural network training by dynamic sparse model reconfiguration," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[216] Y. Xu, X. Dong, Y. Li, and H. Su, "A main/subsidiary network framework for simplifying binary neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7147–7155, 2019.

[217] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4938–4948, 2019.

[218] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[219] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2019.

[220] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, "Robust statistics on riemannian manifolds via the geometric median," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[221] D. Mehta, K. I. Kim, and C. Theobalt, "On implicit filter level sparsity in convolutional neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 520–528, 2019.

[222] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[223] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, jul 2011.

[224] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[225] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 4646–4655, PMLR, 09–15 Jun 2019.

[226] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 925–938, 2019.

[227] J. Zhang, X. Chen, M. Song, and T. Li, "Eager pruning: Algorithm and architecture support for fast training of deep neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, (New York, NY, USA), p. 292–303, Association for Computing Machinery, 2019.

[228] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[229] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[230] X. Xiao, Z. Wang, and S. Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, (Red Hook, NY, USA), Curran Associates Inc., 2019.

[231] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013.

[232] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[233] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 3647–3658, Curran Associates, Inc., 2020.

[234] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[235] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the annual meeting of the cognitive science society*, vol. 33, 2011.

[236] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-ucsd birds 200," Tech. Rep. CNS-TR-201, Caltech, 2010.

[237] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[238] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[239] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition," in *The 2011 international joint conference on neural networks*, pp. 1453–1460, IEEE, 2011.

[240] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *2014 IEEE international conference on image processing (ICIP)*, pp. 343–347, IEEE, 2014.

[241] B. Yaroslav, "Notmnist dataset. technical report," 2011. Online, accessed 15.07.2020.

[242] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, (Red Hook, NY, USA), Curran Associates Inc., 2019.

[243] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, W. Dai, Y. Qi, Y. Chen, W. Lin, and H. Xiong, "Trained rank pruning for efficient deep neural networks," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 14–17, 2019.

[244] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4876–4883, Apr. 2020.

[245] L. Yang, Z. He, and D. Fan, "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 6623–6630, Apr. 2020.

[246] E. Lobacheva, N. Chirkova, A. Markovich, and D. Vetrov, "Structured sparsification of gated recurrent neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4989–4996, Apr. 2020.

[247] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

[248] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, pp. 649–657, 2015.

[249] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, p. 313–330, jun 1993.

[250] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[251] L. Yang, Z. He, Y. Cao, and D. Fan, "Non-uniform dnn structured subnets sampling for dynamic inference," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[252] L. Wang, W. Wu, J. Zhang, H. Liu, G. Bosilca, M. Herlihy, and R. Fonseca, "Fft-based gradient sparsification for the distributed training of deep neural networks," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, (New York, NY, USA), p. 113–124, Association for Computing Machinery, 2020.

[253] A. Labach and S. Valaee, "A framework for neural network pruning using gibbs distributions," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6, 2020.

[254] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li, "Deep neural network acceleration based on low-rank approximated channel pruning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1232–1244, 2020.

[255] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[256] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[257] S.-K. Chao, Z. Wang, Y. Xing, and G. Cheng, "Directional pruning of deep neural networks," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 13986–13998, Curran Associates, Inc., 2020.

[258] S.-K. Chao and G. Cheng, "A generalization of regularized dual averaging and its dynamics," *ArXiv*, vol. abs/1909.10072, 2019.

[259] B. Bartoldson, A. Morcos, A. Barbu, and G. Erlebacher, "The generalization-stability tradeoff in neural network pruning," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 20852–20864, Curran Associates, Inc., 2020.

[260] Z. Wang, F. Li, G. Shi, X. Xie, and F. Wang, "Network pruning using sparse learning and genetic algorithm," *Neurocomputing*, vol. 404, pp. 247–256, 2020.

[261] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv*, vol. abs/1804.02767, 2018.

[262] P. Zhang, Y. Zhong, and X. Li, "Slimyolov3: Narrower, faster and better for real-time uav applications," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[263] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[264] Q. Li, C. Li, and H. Chen, *Incremental Filter Pruning via Random Walk for Accelerating Deep Convolutional Neural Networks*, p. 358–366. New York, NY, USA: Association for Computing Machinery, 2020.

[265] P. Savarese, H. Silva, and M. Maire, "Winning the lottery with continuous sparsification," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 11380–11390, Curran Associates, Inc., 2020.

[266] A. Sahu, A. Dutta, A. M Abdelmoniem, T. Banerjee, M. Canini, and P. Kalnis, "Rethinking gradient sparsification as total error minimization," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[267] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 440–445, Association for Computational Linguistics, Sept. 2017.

[268] D. Alistarh, T. Hoefler, M. Johansson, S. Khirirat, N. Konstantinov, and C. Renggli, "The convergence of sparsified gradient methods," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 5977–5987, Curran Associates Inc., 2018.

[269] A. Dutta, E. H. Bergou, A. M. Abdelmoniem, C. Ho, A. N. Sahu, M. Canini, and P. Kalnis, "On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 3817–3824, AAAI Press, 2020.

[270] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Interspeech*, 2015.

[271] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018.

[272] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[273] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, (Republic and Canton of Geneva, CHE), p. 173–182, International World Wide Web Conferences Steering Committee, 2017.

[274] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS) 5*, vol. 4, no. 19, p. 19, 2015.

[275] J. Ye, S. Zhang, and J. Wang, *Hybrid Network Compression via Meta-Learning*, p. 1423–1431. New York, NY, USA: Association for Computing Machinery, 2021.

[276] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency," in *European Conference on Computer Vision*, pp. 259–277, Springer, 2020.

[277] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 136–144, 2017.

[278] X. Zhou, W. Zhang, Z. Chen, S. Diao, and T. Zhang, "Efficient neural network training via forward and backward propagation sparsification," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[279] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)* (E. R. H. Richard C. Wilson and W. A. P. Smith, eds.), pp. 87.1–87.12, BMVA Press, September 2016.

[280] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3599–3608, June 2021.

[281] Y. Tang, Y. Wang, Y. Xu, Y. Deng, C. Xu, D. Tao, and C. Xu, "Manifold regularized dynamic network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5018–5028, June 2021.

[282] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763, 2017.

[283] D. Gurevin, M. A. Bragin, C. Ding, S. Zhou, L. Pepin, B. Li, and F. Miao, "Enabling retrain-free deep neural network pruning using surrogate lagrangian relaxation," in *IJCAI*, 2021.

[284] M. A. Bragin, P. B. Luh, J. H. Yan, N. Yu, and G. A. Stern, "Convergence of the surrogate lagrangian relaxation method," *Journal of Optimization Theory and applications*, vol. 164, no. 1, pp. 173–201, 2015.

[285] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu, "Only train once: A one-shot neural network training and pruning framework," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[286] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[287] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[288] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[289] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–25, 2019.

[290] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–40, 2017.

[291] L. Lu and Y. Liang, "Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[292] D. Kim, J. Ahn, and S. Yoo, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 1462–1467, 2017.

[293] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ICCAD '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[294] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020.

[295] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International Conference on Machine Learning*, pp. 794–803, 2018.

[296] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in neural information processing systems*, pp. 901–909, 2016.

[297] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, pp. 971–980, 2017.

[298] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[299] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[300] B. Hanin and D. Rolnick, "How to start training: The effect of initialization and architecture," in *Advances in Neural Information Processing Systems*, pp. 571–581, 2018.

[301] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[302] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," 2020.

[303] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.

[304] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 497–514, 2019.

[305] M. Augustin, A. Meinke, and M. Hein, "Adversarial robustness on in-and out-distribution improves explainability," in *European Conference on Computer Vision*, pp. 228–245, Springer, 2020.

[306] J. Bitterwolf, A. Meinke, and M. Hein, "Certifiably adversarially robust detection of out-of-distribution data," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 16085–16095, Curran Associates, Inc., 2020.

[307] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, ch. 7, pp. 119–137. SIAM, 2002.

[308] K. Wang and A. N. Michel, "Robustness and perturbation analysis of a class of artificial neural networks," *Neural Networks*, vol. 7, no. 2, pp. 251–259, 1994.

[309] A. Meyer-Base, "Perturbation analysis of a class of neural networks," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 2, pp. pp. 825–828 vol.2, 1997.

[310] Xiaoqin Zeng and D. S. Yeung, "Sensitivity analysis of multilayer perceptron to input and weight perturbations," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1358–1366, 2001.

[311] L. Xiang, X. Zeng, Y. Niu, and Y. Liu, "Study of sensitivity to weight perturbation for convolution neural network," *IEEE Access*, vol. 7, pp. 93898–93908, 2019.

[312] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8827–8836, 2018.

[313] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[314] A. Shamir, O. Melamed, and O. BenShmuel, "The dimpled manifold model of adversarial examples in machine learning," *arXiv preprint arXiv:2106.10151*, 2021.

[315] D. P. Mandic, "A generalized normalized gradient descent algorithm," *IEEE signal processing letters*, vol. 11, no. 2, pp. 115–118, 2004.

[316] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1310–1318, PMLR, 17–19 Jun 2013.

[317] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[318] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[319] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[320] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[321] L. Zhu, "Thop: Pytorch-opcounter," 2021. Online, accessed 04.04.2022.

[322] JetBrains, "Pycharm - die python-ide für professionelle entwickler." Online, accessed 06.04.2022.

[323] "Github." Online, accessed 06.04.2022.

[324] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[325] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[326] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[327] N. C. Guido van Rossum, Barry Warsaw, "Pep8." `https://www.python.org/dev/peps/pep-0008/`. [Online; accessed January 11, 2020].

[328] N. Corporation, "NVIDIA DGX-1essential instrument for ai research," 2017. Online, accessed 07.07.2021.

[329] S. Development and Support, "Slurm workload manager." Online, accessed 06.04.2022.

[330] e. a. Damian Avila, "Jupyter notebook." `https://jupyter.org/`. [Online; accessed January 10, 2020].

[331] H. Noh, T. You, J. Mun, and B. Han, "Regularizing deep neural networks by noise: Its interpretation and optimization," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[332] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[333] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, "Evolutionary algorithms," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 3, pp. 178–195, 2014.

[334] S. Forrest, "Genetic algorithms," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 77–80, 1996.

[335] D. P. Kroese and R. Y. Rubinstein, "Monte carlo methods," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 1, pp. 48–58, 2012.

[336] J.-Y. Kim, "Chapter five - fpga based neural network accelerators," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning* (S. Kim and G. C. Deka, eds.), vol. 122 of *Advances in Computers*, pp. 135–165, Elsevier, 2021.

[337] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

Acronyms

AC Gradient Accumulation.

ACM Association for Computing Machinery.

ADMM Alternating Direction Method of Multipliers.

ADMMI Alternating Direction Method of Multipliers - Intra-Training.

ADMMR Alternating Direction Method of Multipliers - Retraining.

ADW Adaptive Drop-Weight.

AGS-CL Adaptive Group Sparsity based Continual Learning.

AI Artificial Intelligence.

ANN Artificial Neural Network.

AP AutoPrune.

ASIC Application-Specific Integrated Circuit.

BASE Bielefeld Academic Search Engine.

BN Batch Normalization.

BNN Binary Neural Networks.

C-SGD Centripetal Stochastic Gradient Descent.

CASIA Institute of Automation of Chinese Academy of Sciences.

CF Convolution kernels/Filters.

CIFAR Canadian Institute for Advanced Research.

CNN Convolutional Neural Network.

COCO Common Objects in Context.

CORE Connecting Repositories.

CPU Central Processing Unit.

DK Dynamic K.

DNN Deep Neural Network.

DRACT Dynamically Reconfigurable Add and Collect Tree.

DSR Dynamic Sparse Reparameterization.

EDSR Enhanced Deep Super-Resolution.

EF Efficient Neural Network Training.

EP Eager Pruning.

FBS Feature Boosting and Suppression.

FFT Fast Fourier Transform.

FLOP Floating Point Operation.

FLOPs Floating Point Operations (plural of FLOP, not to be confused with FLOP/s).

FPGA Field Programmable Gate Array.

FPPMO Filter Pruning via Probabilistic Model-based Optimization.

GB Gigabyte.

GCL Guide to Computing Literature.

GD Gradient Diversity.

GDTopK Top-K Gradients with Layerwise Gradient Diversity.

GDTopKMC Monte-Carlo Top-K Gradients with Layerwise Gradient Diversity.

GDTP Greedy Two Dimensional Partition.

GGD Global Gradient Diversity.

GPU Graphics Processing Unit.

gRDA Generalization of Regularized Dual Averages.

GRU Gated Recurrent Units.

GSLRE Global Supervised Low-Rank Expansion.

GSM Global Sparse Momentum.

HSPG Half-Space Stochastic Projected Gradient.

HWDB Handwriting Database.

IDC Integral of Decay Curve.

IEEE Institute of Electrical and Electronics Engineers.

ILGD Intra-Layer Gradient Diversity.

IMDB Internet Movie Database.

ISTA Iterative Shrinkage-Thresholding Algorithm.

KLD Kullbeck-Leibler Divergence.

LAP Low-Rank Approximated Channel Pruning.

LGD Layerwise Gradient Diversity.

LSTM Long Short-Term Memory.

LTH Lottery Ticket Hypothesis.

MAC Multiply-Accumulate.

MAG Magnitude Based Unstructured Pruning.

MC Monte-Carlo.

MLP Multilayer Perceptron.

MNIST Modified National Institute of Standards and Technology.

MSE Mean Squared Error.

NAS Neural Architecture Search.

NCF Neural Collaborative Filtering.

NeurIPS Neural Information Processing Systems.

NIN Network in Network.

NIPS Neural Information Processing Systems.

NN Neural Network.

NP Non-Deterministic Polynomial-Time.

OBD Optimal Brain Damage.

OBS Optimal Brain Surgeon.

OOD Out-of-Distribution.

OOP Object Oriented Programming.

OTO Only Train Once.

P Parameter.

p.p. Percentage points.

PE Processing Element.

PEP Python Enhancement Proposal.

PGD Proximal Gradient Descent.

PR Procrustes.

PSO Particle Swarm Optimization.

PT Prunetrain.

PTB Penn Treebank Dataset.

QAT Quantization Aware Training.

QSGD Qsparse-local-SGD.

RBF Radial Basis Function.

ReLU Recitified Linear Unit.

REP Relative Error Pruning.

RL Reinforcement Learning.

RQ Research Question.

SFP Soft Filter Pruning.

SGD Stochastic Gradient Descent.

SIMD Single Instruction Multiple Data.

SLR Surrogate Lagrangian Relaxation.

SMCD Structured Model Compact Deployment.

SNIP Single Shot Network Pruning based on Connection Sensitivity.

SotA State-of-the-Art.

SSD Single Shot MultiBox Detector.

STE Straight-Through Estimator.

SVD Singular Value Decomposition.

TAA Theory and Application of Algorithms.

TanH Hyperbolic Tangent.

TIMIT Texas Instruments - Massachusetts Institute of Technology.

TRP Trained Rank Pruning.

VCP Variational Convolutional Neural Network Pruning.

VDA Visualization and Data Analysis.

VGG Visual Geometry Group.

VR-PGE Variance Reduced Policy Gradient Estimator.

WL Word Length.

WPC Weights Penalty Clipping.

WRN Wide ResNet.

YOLO You Only Look Once.

Glossary

L Set of all layers l of a neural network, see Section 2.4.2.1.

l Single layer of a neural network, see Section 2.4.2.1.

L_1 L_1 regularization, see Section 2.4.3.2.

L_2 L_2 regularization, see Section 2.4.3.2.

P Some pruning heuristic, see Section 4.1.1.6.

SU Speedup of training, see Section 5.

$\Delta s(\boldsymbol{W})_{lb}$ Global gradient diversity of weights $\boldsymbol{W}$ with lookback lb, see Section 4.1.1.1.

$\Delta s(\boldsymbol{W}^l)_{lb,(h_i,w_i)}$ Intra-layer gradient diversity of weight $\boldsymbol{W}^l$ of linear layer l with lookback lb for indices (h_i, w_i), see Section 4.1.1.4.

$\Delta s(\boldsymbol{W}^l)_{lb,(n_i,c_i,h,w)}$ Intra-layer gradient diversity of weight $\boldsymbol{W}^l$ of convolutional layer l with lookback lb for indices (n_i, c_i, h, w), see Section 4.1.1.4.

$\Delta s(\boldsymbol{W}^l)_{lb}$ Layer-wise gradient diversity of weight $\boldsymbol{W}^l$ of layer l with lookback lb, see Section 4.1.1.2.

Γ^l Kullbeck-leiber divergence based measure of information change in layer l, see Section 4.1.2.2.

Φ Elementwise activation function, see Section 2.3.1.

α Control variable used in L_1 and L_2 regularization, see Section 2.4.3.

$\beta_{(1,2)}$ Coefficients used for computing running averages of gradient and its square variable in Adam, see Section 2.4.2.2.

$\boldsymbol{M}_j^l$ Binary pruning mask of layer l at batch j, see Section 2.6.5.

$\boldsymbol{M}^l$ Binary pruning mask of layer l, see Section 2.6.5.

$\boldsymbol{M}_j$ Set of all binary pruning masks at batch j, see Section 2.6.5.

$\boldsymbol{M}$ Set of all binary pruning masks, see Section 2.6.5.

$\boldsymbol{W}_j^l$ Weight tensor of the l-th layer of a neural network at batch j, see Section 4.1.2.2.

$\boldsymbol{W}^l$ Weight tensor of the l-th layer of a neural network, see Section 2.6.5.

$\boldsymbol{W}_j$ Set of all weights tensors of of a neural network at batch j, see Section 2.4.2.1.

$\boldsymbol{W}$ Set of all weights tensors of of a neural network, see Section 2.4.2.1.

$\boldsymbol{X}_L$ Categorical probability distribution over the layers L of a neural network describing their probability of getting pruned by a pruning heuristic P, see Section 4.1.1.6.

$\boldsymbol{X}_j$ Input tensor of a neural network at batch j, see Section 2.4.2.1.

$\boldsymbol{X}$ Set of all input tensors of a neural network, see Section 2.4.2.1.

$\boldsymbol{Y}$ Set of all targets of a neural network, see Section 2.4.2.1.

$\boldsymbol{p}_L$ Characterizing vector of probabilities used in conjunction with $\boldsymbol{X}_L$, see Section 4.1.1.6.

$\boldsymbol{y}_j$ Target vector (ground-trough) at batch j, see Section 2.4.2.1.

η Learning rate, see Section 2.4.2.1.

$\hat{\mathcal{L}}$ Modified loss function (e.g. with some penalty term added), see Section 2.6.5.

κ_c Quantiles of samples of $\widehat{\Gamma}^l_{n_i,c_i,h,w}$ that are used for mask generation, see Section 4.1.2.4.

κ_l Quantiles of samples of $\widehat{\Gamma}^l_{h_i,w_i}$ that are used for mask generation, see Section 4.1.2.4.

λ Weight decay used in SGD and Adam, see Sections 2.4.2.2 and 2.4.2.1.

$\mathcal{L}$ Loss function, see Section 2.4.1.

$\mathcal{P}$ Some penalty term, see Section 2.6.5.

μ Momentum control variable in SGD, see Section 2.4.2.1.

$\nabla_{\boldsymbol{W}_j}\mathcal{L}$ Gradient of the loss function $\mathcal{L}$ w.r.t the networks weights $\boldsymbol{W}_j$ at batch j, see Section 2.4.2.1.

$\nabla^{(l,acc)}_{\boldsymbol{W}_j}\mathcal{L}$ Accumulated gradient of the loss function $\mathcal{L}$ w.r.t the networks weights $\boldsymbol{W}_j$ at batch j, see Section 4.1.2.4.

ω Growth factor used in the dynamic adjustment of ac and k, see Section 4.2.4.

σ_c Scaling parameter used to determine indices n_i and c_i in $\widehat{\Gamma}^l_{n_i,c_i,h,w}$, see Section 4.1.2.4.

σ_l Scaling parameter used to determine indices h_i and w_i in $\widehat{\Gamma}^l_{h_i,w_i}$, see Section 4.1.2.4.

τ Dampening control variable in SGD, see Section 2.4.2.1.

$\widehat{\Gamma}^l$ Approximation of Γ^l for layer l, see Section 4.1.2.3.

$\widehat{\Gamma}^l_{h_i,w_i}$ Intra-layer formulation of the approximation of Γ^l for linear layer l with indices h_i, w_i, see Section 4.1.2.3.

$\widehat{\Gamma}^l_{n_i,c_i,h,w}$ Intra-layer formulation of the approximation of Γ^l for convolutional layer l with indices n_i, c_i, h, w, see Section 4.1.2.3.

$\widehat{\nabla}_{\boldsymbol{W}_j}\mathcal{L}$ Modified (e.g. normalized) gradient of the loss function $\mathcal{L}$ w.r.t the networks weights $\boldsymbol{W}_j$ at batch j, see Section 4.1.1.3.

ζ_c Number of samples to be computed with $\widehat{\Gamma}^l_{n_i,c_i,h,w}$, see Section 4.1.2.4.

ζ_l Number of samples to be computed with $\widehat{\Gamma}^l_{h_i,w_i}$, see Section 4.1.2.4.

ac Number of gradients to accumulate before triggering a weight update step, see Section 4.2.2.

$flops^l$ FLOPs incurred during a forward pass in layer l, see Section 5.

$flops^l_{bwd,j}$ FLOPs incurred during a backwards pass with in layer l at batch j adjusted for sparsity sp^l_j and $sp^l_{grad,j}$, see Section 5.

$flops^l_{fwd,j}$ FLOPs incurred during a forward pass in layer l at batch j adjusted for sparsity sp^l_j, see Section 5.

$flops_{total}$ Total FLOPs incurred during training, see Section 5.

k Number of gradients to delete in top-k gradients, see Section 4.2.4.

lb Lookback variable controlling the number of gradients used gradient diversity and pruning heuristic $\widehat{\Gamma}^l$, see Section 4.1.1.1 and 4.1.2.4.

$oh^l_{ac,j}$ Algorithmic overhead incurred through gradient accumulation in layer l batch j, see Section 5.

$oh^l_{admm,j}$ Algorithmic overhead incurred through ADMM in layer l batch j, see Section 5.

$oh^l_{app,j}$ Algorithmic overhead incurred through mask application in layer l batch j, see Section 5.

$oh^l_{mask,j}$ Algorithmic overhead incurred through REP mask generation in layer l batch j, see Section 5.

$oh^l_{search,j}$ Algorithmic overhead incurred through REP search in layer l batch j, see Section 5.

$oh^l_{topk,j}$ Algorithmic overhead incurred through GDTopK in layer l batch j, see Section 5.

oh Total algorithmic overhead of applying the proposed pruning approaches, see Section 5.

pe Number of epochs that the pruning heuristic is active, see Section 4.1.2.4.

sp_j^l Weights sparsity observed in layer l at batch j, see Section 5.

$sp_{grad,j}^l$ Gradient sparsity observed in layer l at batch j, see Section 5.

Ablation study Analytically or empirically executed study of the different components of algorithm with the goal to gain insight into the contribution of these components to the achieving of the algorithms goal.

Application-specific integrated circuit An integrated circuit which is purpose-built and optimized towards a specific application instead of general use.

Architecture Strictly speaking, the architecture of an artificial neural network describes the way in which layers are stacked and what types of layers are used [1]. In literature though, this term is often used synonymously with topology and this use is also employed in this thesis.

Artificial neural network An artificial neural network is a machine learning method simulating the learning mechanism of certain parts of the human nervous system [2, p. 1].

Axon Part of the connection between neurons [2, p. 1].

Backpropagation The backpropagation algorithm uses the chain rule to compute the error gradients in terms of summations of local-gradient products over the various paths from a node to the output [2, p. 21].

Backwards pass In the backwards pass, the gradients of the loss function with respect to the networks weights are computed using chain rule and the weights are subsequently updated using these gradients [2, p. 21].

Baseline An unmodified reference model against which a modified model (i.e. a pruned one in the context of this thesis) is compared in order to study the effects of the modifications.

Convergence Artificial neural network training is said to have converged if additional iterations do not change the solution significantly. The term convergence speed describes the rate at which an artificial neural network trained by some iterative optimization algorithm moves towards this optimum i.e. the (local) minimum the loss function.

Dendrite Part of the connection between neurons [2, p. 1].

Evolutionary algorithm describes population-based search algorithms that are motivated natural evolution [333].

Failt-injection Injection of e.g. bit-faults into the weights of an artificial neural network [54].

Field programmable gate array An integrated circuit which can be modified by a user after production. Often used in prototypical realizations of new integrated circuits.

Floating-point operation A computation step in floating-point precision.

Forwards pass In the forwards pass, the inputs for a training data point are fed into the network, leading to a forward cascade of computations through the networks layers, resulting in a predicted output [2, p. 21].

Generalization error Error (measured e.g. in the percentage of falsely classified images in an image classification task) of a neural network on data other than the training data, i.e. unseen data [2, p. 172].

Genetic algorithm A computational model of biological evolution. Genetic algorithms are used to solve search problems as well as modeling evolutionary systems. [334].

Gradient exploding During backpropagation, the updates in earlier layers can either be very small (vanishing gradient) or very large (exploding gradient) in certain types of artificial neural networks [2, p. 28].

Gradient vanishing During backpropagation, the updates in earlier layers can either be very small (vanishing gradient) or very large (exploding gradient) in certain types of artificial neural networks [2, p. 28].

Hessian Second order derivative of the loss function with respect to the artificial neural networks weights [2, p. 143].

Hyperparamater The term hyperparameter refers to the parameters regulating the design of the model (like e.g. learning rate),. Hyperparameters are different from the parameters representing the weights of connections in the artificial neural network [2, p. 125].

Interpretability The interpretability of an artificial neural network refers to the degree humans can understand its their black-box representations.

Monte carlo methods Solving quantitative problems through statistical sampling, e.g. to estimate numerical quantities by repeated sampling [335].

Multiply-accumulate A computation step that multiplies to factors and adds a third number to the product.

Neuron Certain cell type of the human nervous system [2, p. 1].

Overfitting Describes the process of training an artificial neural network in such a fashion that it has a very low error on the training data but a high error on unseen data (i.e. a high generalization error) [2, p. 25].

Processing element A basic compute unit, usually consists of a MAC unit and register-/scratchpad memories [336].

Pseudo code Illustrative description of the steps of an algorithm.

PyTorch A popular machine learning framework [56].

Reinforcement learning handles learning in sequential decision making problems where onyl limited feedback is available [337].

Representational power The ability of a neural network to learn certain concepts or functions [1].

Robustness The robustness of an artificial neural network refers to the degree it can resist accuracy degradation in the face of adverserial attacks on or perturbations of its weights or input data.

Synapse Connecting region between dendrites and axons [2, p. 1].

TensorFlow A popular machine learning framework [57].

Topology Strictly speaking, the topology of an artificial neural network describes the way in which the artificial neurons in the network are connected [1]. In literature though, this term is often used synonymously with architecture and this use is also employed in this thesis.

A. Appendix

A.1. Technical Documentation

This section contains instructions for installation, execution and reproduction of results as well as annotated *example* configs for each of the experiments conducted during the evaluation of this thesis. For the configs actually used, please refer to the GitHub repository[1].

A.1.1. Installation, Execution and Reproduction of Results

First, clone GitHub repository.

```
1    git clone https://github.com/lorenz0890/multi-directional-pruning.git
```

Listing A.1: Command for cloning the code repository.

Next, install requirements:

```
1    pip install -r requirements.txt
```

Listing A.2: Command for installing the necessary software dependencies.

Now, two modes of operation are available: individual config mode and batch mode. In the first case, a path to a specific config must be provided and the associated experiment:

```
1 python main.py --config /path/to/a/single/config.json
```

Listing A.3: Command for running a single experiment.

In the second case, a path to a directory containing config files must be provided:

```
1 python main.py --batch /path/to/directory/of/multiple/configs/
```

Listing A.4: Command for running multiple experiments.

For details on the structures of the configs associated with different experiments as well the adjustable parameters, see Appendix A. The configs necessary to produce the main results of this thesis will be provided along with the code base in the accompanying GitHub repository.

A.1.2. Annotated Example Configs

[1] Link: `https://github.com/lorenz0890/multi-directional-pruning`.

```
[EXPERIMENT]
name = admm_intra #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 1 #number of pre-training epochs
epochs = 1 #number of admm-training epochs
re_epochs = 1 #number of re-training epochs
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam
repeat = 3 #repetitions of ADMMI

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.5: Example config for ADMMI.

```
[EXPERIMENT]
name = admm_retrain #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 3 #number of pre-training epochs
epochs = 10 #number of admm-training epochs
re_epochs = 3 #number of re-training epochs
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
```

```
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.6: Example config for ADMMR.

```
[EXPERIMENT]
name = gd_top_k #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
epochs = 10 #number of epochs
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.7: Example config for GDTopK.

```
[EXPERIMENT]
name = gd_top_k_mc #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3 #sampling epochs
epochs = 10 #number of epochs
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

[OTHER]
no_cuda = False #use GPU/CPU (not used)
```

```
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.8: Example config for GDTopKMC.

```
[EXPERIMENT]
name = gd_top_k_mc_ac #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3 #sampling epochs
ac = 2 #gradient accumulation
epochs = 10 #number of epochs
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.9: Example config for GDTopKMC+AC.

```
[EXPERIMENT]
name = gd_top_k_mc_ac_dk #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
ac = 8 #gradient accumulation
epochs = 10 #number of epochs
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
lr = 1e-3 #learning rate
```

```
adam_eps = 1e-8 #eps if optimizer is adam

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.10: Example config for GDTopKMC+AC+DK.

```
[EXPERIMENT]
name = gd_top_k_mc_ac_dk_admm_intra #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
ac = 8 #gradient accumulation
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 1 #number of pre-training epochs
epochs = 1 #number of admm-training epochs
re_epochs = 1 #number of re-training epochs
repeat = 3 #repetitions of ADMMI

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.11: Example config for GDTopKMC+AC+DK+ADMMI.

```
[EXPERIMENT]
name = gd_top_k_mc_ac_dk_admm_retrain #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size
```

```

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
ac = 8 #gradient accumulation
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 2 #number of pre-training epochs
epochs = 6 #number of admm-training epochs
re_epochs = 2 #number of re-training epochs

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.12: Example config for GDTopKMC+AC+DK+ADMMR.

```
[EXPERIMENT]
name = re_pruning #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
lb = 50 #look back (batches)
num_classes = 10 #number of classes
epochs = 10 #number of epochs
lr = 1e-1 #learning rate
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
steps = 5, 7 #learning rate scheduling steps (epochs)
gamma = 0.1 #learning rate scheduling reduction at step

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
```

```
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.13: Example config for REP.

```
[EXPERIMENT]
name = re_pruning_ac #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
lb = 50 #look back (batches)
num_classes = 10 #number of classes
epochs = 10 #number of epochs
lr = 1e-1 #learning rate
l1 = 1e-4 #strength of l1 regularization
l2 = 1e-4 #strength of l2 regularization
steps = 5, 7 #learning rate scheduling steps (epochs)
gamma = 0.1 #learning rate scheduling reduction at step

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

ac = 2 #gradient accumulation

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
```

```
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.14: Example config for REP+AC.

```
[EXPERIMENT]
name = re_pruning_ac_admm_intra #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 1 #number of pre-training epochs
epochs = 1 #number of admm-training epochs
re_epochs = 1 #number of re-training epochs
repeat = 3 #repetitions of ADMMI

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

ac = 2 #gradient accumulation

[OTHER]
no_cuda = False
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.15: Example config for REP+AC+ADMMI.

```
[EXPERIMENT]
```

```
name = re_pruning_ac_admm_retrain #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 2 #number of pre-training epochs
epochs = 6 #number of admm-training epochs
re_epochs = 2 #number of re-training epochs

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

ac = 2 #gradient accumulation

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.16: Example config for REP+AC+ADMMR.

```
[EXPERIMENT]
name = re_pruning_admm_intra #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size
```

```

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
ac = 8 #gradient accumulation
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 1 #number of pre-training epochs
epochs = 1 #number of admm-training epochs
re_epochs = 1 #number of re-training epochs
repeat = 3 #repetitions of ADMMI

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.17: Example config for REP+ADMMI.

```
[EXPERIMENT]
name = re_pruning_admm_retrain #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

```

```
percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 2 #number of pre-training epochs
epochs = 6 #number of admm-training epochs
re_epochs = 2 #number of re-training epochs

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.18: Example config for REP+ADMMR.

```
[EXPERIMENT]
name = re_pruning_gd_top_k_mc_ac_dk_admm_intra #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
ac = 8 #gradient accumulation
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 1 #number of pre-training epochs
```

```
epochs = 1 #number of admm-training epochs
re_epochs = 1 #number of re-training epochs
repeat = 3 #repetitions of ADMMI

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.19: Example config for REP+GDTopKMC+AC+DK+ADMMI.

```
[EXPERIMENT]
name = re_pruning_gd_top_k_mc_ac_dk_admm_retrain #name of experiment
model = lenet #name of model
dataset = mnist #name of dataset
train_batch_size = 64 #training batch size
test_batch_size = 1000 #testing batch size

[SPECIFICATION]
num_classes = 10 #number of classes
lb = 50 #look back (batches)
k = 3 #number of gradients to prune
se = 3  #sampling epochs
ac = 8 #gradient accumulation
lr = 1e-3 #learning rate
adam_eps = 1e-8 #eps if optimizer is adam

percent = 0.8, 0.92, 0.991, 0.93 #per-layer pruning percentages in ADMM
alpha = 5e-4 #ADMM-specific hyperparameter
rho = 1e-2 #ADMM-specific hyperparameter
l1 = False #use l1 for pruning in ADMM
l2 = False #use l2 for pruning in ADMM
pre_epochs = 2 #number of pre-training epochs
epochs = 6 #number of admm-training epochs
re_epochs = 2 #number of re-training epochs

softness_c = 0.0 #softness of pruning mask application
magnitude_t_c = 1e-3 #magnitude pruning threshold
```

```
metric_q_c = 0.25 #quantile of blocks to prune
sample_c = 1000 #number of samples
scale_c = 0.01 #scaling factor for pruning block search

softness_l = 0.0 #softness of pruning mask application
magnitude_t_l = 1e-3  #magnitude pruning threshold
metric_q_l = 0.05 #quantile of blocks to prune
sample_l = 1000 #number of samples
scale_l = 0.1 #scaling factor for pruning block search
prune_epochs = 0 #epochs to prune (0 = first epoch)

[OTHER]
no_cuda = False #use GPU/CPU (not used)
seed = 1 #set random seed for reproduceability
save_model = False #save model after training (not used)
vis_model = False #visualize model layers after training
vis_log = True #visualize log file after training
out_path = /path/to/logfiles #location to store logfiles
```

Listing A.20: Example config for REP+GDTopKMC+AC+DK+ADMMR.

A.2. Additional Tables

Paramater	Count	Mean	Std	Min	Max
lr	$3.0 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.21 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
pre_epochs	$3.0 \cdot 10^{1}$	1.40	$4.98 \cdot 10^{-1}$	1.0	2.0
epochs	$3.0 \cdot 10^{1}$	1.40	$4.98 \cdot 10^{-1}$	1.0	2.0
re_epochs	$3.0 \cdot 10^{1}$	1.40	$4.98 \cdot 10^{-1}$	1.0	2.0
repeat	$3.0 \cdot 10^{1}$	2.40	$4.98 \cdot 10^{-1}$	2.0	3.0
train_batch_size	$3.0 \cdot 10^{1}$	$1.6 \cdot 10^{2}$	$9.76 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.1.: Hyperparameter range for ADMMI

Parameter	Count	Mean	Std	Min	Max
lr	$3.0 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.21 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
pre_epochs	$3.0 \cdot 10^{1}$	4.80	2.76	3.0	$1.0 \cdot 10^{1}$
epochs	$3.0 \cdot 10^{1}$	6.40	3.19	3.0	$1.0 \cdot 10^{1}$
re_epochs	$3.0 \cdot 10^{1}$	4.80	2.76	3.0	$1.0 \cdot 10^{1}$
train_batch_size	$3.0 \cdot 10^{1}$	$1.6 \cdot 10^{2}$	$9.76 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.2.: Hyperparameter range for ADMMR

Parameter	Count	Mean	Std	Min	Max
lr	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$1.5 \cdot 10^{1}$	3.0	0.0	3.0	3.0
train_batch_size	$1.5 \cdot 10^{1}$	$1.6 \cdot 10^{2}$	$9.91 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.3.: Hyperparameter range for GDTopK

Parameter	Count	Mean	Std	Min	Max
lr	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$4.49 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$1.5 \cdot 10^{1}$	3.0	0.0	3.0	3.0
se	$1.5 \cdot 10^{1}$	2.53	$5.16 \cdot 10^{-1}$	2.0	3.0
train_batch_size	$1.5 \cdot 10^{1}$	$1.66 \cdot 10^{2}$	$9.91 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.4.: Hyperparameter range for GDTopKMC

Parameter	Count	Mean	Std	Min	Max
lr	$3.8 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.20 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$3.8 \cdot 10^{1}$	2.68	$7.39 \cdot 10^{-1}$	1.0	3.0
se	$3.8 \cdot 10^{1}$	2.79	$4.13 \cdot 10^{-1}$	2.0	3.0
ac	$3.8 \cdot 10^{1}$	6.74	3.85	2.0	$1.20 \cdot 10^{1}$
train_batch_size	$3.8 \cdot 10^{1}$	$1.25 \cdot 10^{2}$	$9.04 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.5.: Hyperparameter range for GDTopKMC+AC

Parameter	Count	Mean	Std	Min	Max
lr	$4.2 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$4.2 \cdot 10^{1}$	2.52	1.02	1.0	4.0
se	$4.2 \cdot 10^{1}$	2.71	$4.57 \cdot 10^{-1}$	2.0	3.0
ac	$4.2 \cdot 10^{1}$	7.14	3.03	2.0	$1.20 \cdot 10^{1}$
train_batch_size	$4.2 \cdot 10^{1}$	$1.33 \cdot 10^{2}$	$9.31 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.6.: Hyperparameter range for GDTopKMC+AC+DK

Parameter	Count	Mean	Std	Min	Max
lr	$3.6 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$3.6 \cdot 10^{1}$	2.56	$8.43 \cdot 10^{-1}$	1.0	3.0
se	$3.6 \cdot 10^{1}$	2.78	$4.22 \cdot 10^{-1}$	2.0	3.0
ac	$3.6 \cdot 10^{1}$	7.0	3.26	2.0	$1.20 \cdot 10^{1}$
pre_epochs	$3.6 \cdot 10^{1}$	1.33	$4.78 \cdot 10^{-1}$	1.0	2.0
epochs	$3.6 \cdot 10^{1}$	1.33	$4.78 \cdot 10^{-1}$	1.0	2.0
re_epochs	$3.6 \cdot 10^{1}$	1.67	$7.56 \cdot 10^{-1}$	1.0	3.0
repeat	$3.6 \cdot 10^{1}$	2.33	$4.78 \cdot 10^{-1}$	2.0	3.0
train_batch_size	$3.6 \cdot 10^{1}$	$1.28 \cdot 10^{2}$	$9.18 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.7.: Hyperparameter range for GDTopKMC+AC+DK+ADMMI

Parameter	Count	Mean	Std	Min	Max
lr	$3.3 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
k	$3.3 \cdot 10^{1}$	2.64	$7.83 \cdot 10^{-1}$	1.0	3.0
se	$3.3 \cdot 10^{1}$	2.76	$4.35 \cdot 10^{-1}$	2.0	3.0
ac	$3.3 \cdot 10^{1}$	7.27	3.27	2.0	$1.20 \cdot 10^{1}$
pre_epochs	$3.3 \cdot 10^{1}$	3.27	2.53	1.0	$1.0 \cdot 10^{1}$
epochs	$3.3 \cdot 10^{1}$	4.27	2.04	1.0	6.0
re_epochs	$3.3 \cdot 10^{1}$	6.27	4.69	2.0	$1.40 \cdot 10^{1}$
train_batch_size	$3.3 \cdot 10^{1}$	$1.34 \cdot 10^{2}$	$9.38 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.8.: Hyperparameter range for GDTopKMC+AC+DK+ADMMR

Parameter	Count	Mean	Std	Min	Max
lr	$1.0 \cdot 10^{1}$	$1.0 \cdot 10^{-1}$	0.0	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$
prune_epochs	$1.0 \cdot 10^{1}$	$4.0 \cdot 10^{-1}$	$8.43 \cdot 10^{-1}$	0.0	2.0
metric_q_l	$1.0 \cdot 10^{1}$	$3.52 \cdot 10^{-2}$	$2.07 \cdot 10^{-2}$	$1.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-2}$
metric_q_c	$1.0 \cdot 10^{1}$	$1.85 \cdot 10^{-1}$	$8.76 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-1}$
scale_l	$1.0 \cdot 10^{1}$	$1.70 \cdot 10^{-1}$	$7.53 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$1.0 \cdot 10^{1}$	$7.60 \cdot 10^{-2}$	$9.88 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-1}$
sample_l	$1.0 \cdot 10^{1}$	$8.2 \cdot 10^{2}$	$3.79 \cdot 10^{2}$	$1.0 \cdot 10^{2}$	$1.0 \cdot 10^{3}$
sample_c	$1.0 \cdot 10^{1}$	$8.2 \cdot 10^{2}$	$3.79 \cdot 10^{2}$	$1.0 \cdot 10^{2}$	$1.0 \cdot 10^{3}$
softness_l	$1.0 \cdot 10^{1}$	$1.80 \cdot 10^{-1}$	$3.79 \cdot 10^{-1}$	0.0	$9.0 \cdot 10^{-1}$
softness_c	$1.0 \cdot 10^{1}$	$1.80 \cdot 10^{-1}$	$3.79 \cdot 10^{-1}$	0.0	$9.0 \cdot 10^{-1}$
magnitude_t_c	$1.0 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.29 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$1.0 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.29 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
l1	$1.0 \cdot 10^{1}$	$1.0 \cdot 10^{-4}$	0.0	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
l2	$1.0 \cdot 10^{1}$	$8.00 \cdot 10^{-5}$	$4.22 \cdot 10^{-5}$	$1.00 \cdot 10^{-8}$	$1.0 \cdot 10^{-4}$
train_batch_size	$1.0 \cdot 10^{1}$	$1.6 \cdot 10^{2}$	$1.01 \cdot 10^{2}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.9.: Hyperparameter range for REP

Parameter	Count	Mean	Std	Min	Max
lr	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{-1}$	0.0	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$
prune_epochs	$1.5 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$1.5 \cdot 10^{1}$	$4.38 \cdot 10^{-2}$	$1.12 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$1.5 \cdot 10^{1}$	$2.19 \cdot 10^{-1}$	$5.59 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$1.5 \cdot 10^{1}$	$1.38 \cdot 10^{-1}$	$6.71 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$1.5 \cdot 10^{1}$	$3.25 \cdot 10^{-2}$	$4.02 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
sample_l	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	$1.5 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$1.5 \cdot 10^{1}$	0.0	0.0	0.0	0.0
magnitude_t_c	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$1.5 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
l1	$1.5 \cdot 10^{1}$	$1.56 \cdot 10^{-4}$	$2.25 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-3}$
l2	$1.5 \cdot 10^{1}$	$1.56 \cdot 10^{-4}$	$2.25 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-3}$
ac	$1.5 \cdot 10^{1}$	3.12	1.63	2.0	8.0
train_batch_size	$1.5 \cdot 10^{1}$	$1.72 \cdot 10^{2}$	$9.84 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.10.: Hyperparameter range for REP+AC

Parameter	Count	Mean	Std	Min	Max
lr	$4.8 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
prune_epochs	$4.8 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$4.8 \cdot 10^{1}$	$4.11 \cdot 10^{-2}$	$1.21 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$4.8 \cdot 10^{1}$	$2.06 \cdot 10^{-1}$	$6.04 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$4.8 \cdot 10^{1}$	$1.41 \cdot 10^{-1}$	$6.74 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$4.8 \cdot 10^{1}$	$3.44 \cdot 10^{-2}$	$4.04 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
sample_l	$4.8 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	$4.8 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	$4.8 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$4.8 \cdot 10^{1}$	0.0	0.0	0.0	0.0
magnitude_t_c	$4.8 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$4.8 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
repeat	$4.8 \cdot 10^{1}$	2.46	$5.04 \cdot 10^{-1}$	2.0	3.0
pre_epochs	$4.8 \cdot 10^{1}$	1.40	$4.94 \cdot 10^{-1}$	1.0	2.0
epochs	$4.8 \cdot 10^{1}$	1.31	$4.68 \cdot 10^{-1}$	1.0	2.0
re_epochs	$4.8 \cdot 10^{1}$	1.38	$4.89 \cdot 10^{-1}$	1.0	2.0
train_batch_size	$4.8 \cdot 10^{1}$	$1.84 \cdot 10^{2}$	$9.39 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.11.: Hyperparameter range for REP+ADMMI

Parameter	Count	Mean	Std	Min	Max
lr	$6.1 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
prune_epochs	$6.1 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$6.1 \cdot 10^{1}$	$3.73 \cdot 10^{-2}$	$1.26 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$6.1 \cdot 10^{1}$	$1.86 \cdot 10^{-1}$	$6.30 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$6.1 \cdot 10^{1}$	$1.37 \cdot 10^{-1}$	$6.51 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$6.1 \cdot 10^{1}$	$3.21 \cdot 10^{-2}$	$3.91 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
sample_l	$6.1 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	$6.1 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	$6.1 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$6.1 \cdot 10^{1}$	0.0	0.0	0.0	0.0
magnitude_t_c	$6.1 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$6.1 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
pre_epochs	$6.1 \cdot 10^{1}$	2.51	1.77	1.0	6.0
epochs	$6.1 \cdot 10^{1}$	2.57	1.82	1.0	6.0
re_epochs	$6.1 \cdot 10^{1}$	2.62	1.83	1.0	6.0
train_batch_size	$6.1 \cdot 10^{1}$	$1.99 \cdot 10^{2}$	$8.83 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.12.: Hyperparameter range for REP+ADMMR

Parameter	Count	Mean	Std	Min	Max
lr	$7.6 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.18 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
prune_epochs	$7.6 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$7.6 \cdot 10^{1}$	$3.98 \cdot 10^{-2}$	$1.24 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$7.6 \cdot 10^{1}$	$1.99 \cdot 10^{-1}$	$6.18 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$7.6 \cdot 10^{1}$	$1.45 \cdot 10^{-1}$	$6.94 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$7.6 \cdot 10^{1}$	$3.72 \cdot 10^{-2}$	$4.16 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
sample_l	$7.6 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	$7.6 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	$7.6 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$7.6 \cdot 10^{1}$	0.0	0.0	0.0	0.0
magnitude_t_c	$7.6 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.18 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$7.6 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.18 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
ac	$7.6 \cdot 10^{1}$	3.16	$9.94 \cdot 10^{-1}$	2.0	4.0
repeat	$7.6 \cdot 10^{1}$	2.61	$4.92 \cdot 10^{-1}$	2.0	3.0
train_batch_size	$7.6 \cdot 10^{1}$	$1.95 \cdot 10^{2}$	$8.98 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.13.: Hyperparameter range for REP+AC+ADMMI

Parameter	Count	Mean	Std	Min	Max
lr	$4.7 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.19 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
prune_epochs	$4.7 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$4.7 \cdot 10^{1}$	$4.15 \cdot 10^{-2}$	$1.20 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$4.7 \cdot 10^{1}$	$2.07 \cdot 10^{-1}$	$5.99 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$4.7 \cdot 10^{1}$	$1.41 \cdot 10^{-1}$	$6.78 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_c	$4.7 \cdot 10^{1}$	$3.49 \cdot 10^{-2}$	$4.07 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
sample_l	$4.7 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	$4.7 \cdot 10^{1}$	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	$4.7 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$4.7 \cdot 10^{1}$	0.0	0.0	0.0	0.0
magnitude_t_c	$4.7 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.19 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$4.7 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.19 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
ac	$4.7 \cdot 10^{1}$	3.02	1.01	2.0	4.0
pre_epochs	$4.7 \cdot 10^{1}$	3.17	1.71	2.0	6.0
epochs	$4.7 \cdot 10^{1}$	3.60	1.86	2.0	6.0
re_epochs	$4.7 \cdot 10^{1}$	3.23	1.70	2.0	6.0
train_batch_size	$4.7 \cdot 10^{1}$	$1.87 \cdot 10^{2}$	$9.33 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.14.: Hyperparameter range for REP+AC+ADMMR

Parameter	Count	Mean	Std	Min	Max
lr	$1.1 \cdot 10^{1}$	$9.10 \cdot 10^{-2}$	$2.98 \cdot 10^{-2}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-1}$
prune_epochs	$1.1 \cdot 10^{1}$	0.0	0.0	0.0	0.0
metric_q_l	$1.1 \cdot 10^{1}$	$5.0 \cdot 10^{-2}$	0.0	$5.0 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	$1.1 \cdot 10^{1}$	$2.5 \cdot 10^{-1}$	0.0	$2.5 \cdot 10^{-1}$	$2.5 \cdot 10^{-1}$
scale_l	$1.1 \cdot 10^{1}$	$1.0 \cdot 10^{-1}$	0.0	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$
scale_c	$1.1 \cdot 10^{1}$	$1.0 \cdot 10^{-2}$	$1.82 \cdot 10^{-18}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$
sample_l	$1.1 \cdot 10^{1}$	$9.18 \cdot 10^{2}$	$2.71 \cdot 10^{2}$	$1.0 \cdot 10^{2}$	$1.0 \cdot 10^{3}$
sample_c	$1.1 \cdot 10^{1}$	$9.18 \cdot 10^{2}$	$2.71 \cdot 10^{2}$	$1.0 \cdot 10^{2}$	$1.0 \cdot 10^{3}$
softness_l	$1.1 \cdot 10^{1}$	0.0	0.0	0.0	0.0
softness_c	$1.1 \cdot 10^{1}$	$1.64 \cdot 10^{-1}$	$3.64 \cdot 10^{-1}$	0.0	$9.0 \cdot 10^{-1}$
magnitude_t_c	$1.1 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.27 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	$1.1 \cdot 10^{1}$	$1.0 \cdot 10^{-3}$	$2.27 \cdot 10^{-19}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
ac	$1.1 \cdot 10^{1}$	4.55	1.81	2.0	8.0
train_batch_size	$1.1 \cdot 10^{1}$	$2.04 \cdot 10^{2}$	$8.97 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.15.: Hyperparameter range for REP+GDTopKMC+AC+DK

Parameter	Count	Mean	Std	Min	Max
lr	7.0	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
prune_epochs	7.0	0.0	0.0	0.0	0.0
metric_q_l	7.0	$5.0 \cdot 10^{-2}$	$7.49 \cdot 10^{-18}$	$5.0 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
metric_q_c	7.0	$5.0 \cdot 10^{-2}$	$7.49 \cdot 10^{-18}$	$5.0 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
scale_l	7.0	$1.0 \cdot 10^{-1}$	$1.50 \cdot 10^{-17}$	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$
scale_c	7.0	$1.0 \cdot 10^{-2}$	0.0	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$
sample_l	7.0	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
sample_c	7.0	$1.0 \cdot 10^{3}$	0.0	$1.0 \cdot 10^{3}$	$1.0 \cdot 10^{3}$
softness_l	7.0	$6.43 \cdot 10^{-1}$	$4.39 \cdot 10^{-1}$	0.0	$9.0 \cdot 10^{-1}$
softness_c	7.0	$6.43 \cdot 10^{-1}$	$4.39 \cdot 10^{-1}$	0.0	$9.0 \cdot 10^{-1}$
magnitude_t_c	7.0	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
magnitude_t_l	7.0	$1.0 \cdot 10^{-3}$	0.0	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
ac	7.0	6.29	2.14	4.0	8.0
train_batch_size	7.0	$2.01 \cdot 10^{2}$	$9.37 \cdot 10^{1}$	$6.4 \cdot 10^{1}$	$2.56 \cdot 10^{2}$

Table A.16.: Hyperparameter range for REP+GDTopKMC+AC+DK+ADMMR

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
VCP	CIFAR-10	DenseNet-40	−0.95 p.p.	60.00%	59.67%	44.78%	?
		ResNet-56	−0.78 p.p.	45.00%	20.49%	20.30%	?
		ResNet-110	−0.26 p.p.	63.00%	41.27%	36.44%	?
		ResNet-164	−0.42 p.p.	74.00%	56.70%	49.08%	?
	CIFAR-100	DensNet-40	−2.45 p.p.	37.00%	37.73%	22.67%	?
		ResNet-164	−1.80 p.p.	47.00%	17.59%	27.16%	?
	ImageNet	ResNet-50	+0.10 p.p	40.00%	?	?	?
SNIP	CIFAR-10	VGG-like	+0.26 p.p	?	97.00%	?	?
		WRN-16-8	−0.42 p.p.	?	95.00%	?	?
		WRN-16-10	−0.52 p.p.	?	95.00%	?	?
		WRN-22-8	+0.29 p.p.	?	95.00%	?	?
	MNIST	LSTM-s	+0.31 p.p.	?	95.00%	?	?
		LSTM-b	−0.20 p.p.	?	95.00%	?	?
		GRU-s	−0.54 p.p	?	95.00%	?	?
		GRU-b	+0.19 p.p	?	95.00%	?	?
DSR	ImageNet	ResNet-50	−1.60 p.p.	?	90.00%	?	?
			−3.30 p.p.	?	80.00%	?	?
	CIFAR-10	WRN-28-2	?	?	90.00%	?	?
			?	?	80.00%	?	?
AP	ImageNet	AlexNet	−0.84 p.p	?	94.59%	?	?
		ResNet-50	−0.40 p.p	?	54.54%	?	?
LTH	CIFAR-10	Conv-2	?	10.00%	19.91%	?	?
		Conv-4	?	10.00%	18.92%	?	?
		Conv-6	?	15.00%	16.77%	?	?
		VGG-19	?	20.00%	20.00%	?	?
GSM	CIFAR-10	ResNet-56	+0.05 p.p.	?	85.00%	?	?
			−0.25 p.p.	?	90.00%	?	?
		DenseNet-40	+0.21 p.p.	?	85.00%	?	?
			+0.16 p.p.	?	87.50%	?	?
			+0.04 p.p.	?	90.00%	?	?
	ImageNet	ResNet-50	−0.39 p.p.	?	75.00%	?	?
			−1.42 p.p.	?	80.00%	?	?

Table A.17.: Overview of the performance of works with a possible effect during training but not documented in original paper with comparable experiments missing in this thesis. Fields with % read as reduced by X%. LeNet-A=LeNet-300-100. Abbreviations: GSM = Global Sparse Momentum SGD, LTH = Lottery Ticket Hypothesis, DSR = Dynamic Sparse Reparameterization, VCP = Variational Pruning, SNIP = SNIP, AP = AutoPrune, see Table 3.5, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

Paper	Data	Model	Δ Acc.	C/F.	P.	FLOPs(I)	FLOPs(T)
EP	ImageNet	AlexNet	−0.10 p.p.	?	61.69%	?	41.19%
		GoogleNet	−0.80 p.p.	?	70.85%	?	43.25%
		NIN	−1.16 p.p.	?	35.90%	?	26.87%
		ResNet-50	−0.27 p.p.	?	57.81%	?	40.13%
	TIMIT	LSTM	−1.71 p.p.	?	72.07%	?	52.40%
PT	ImageNet	ResNet-50	−1.87 p.p.	?	?	40.00%	53.00%
			−1.47 p.p.	?	?	30.00%	44.00%
			−0.24 p.p.	?	?	3.00%	12.00%
DP	CIFAR-10	DenseNet	+0.62 p.p.	?	77.78%[4]	?	?
			−2.94 p.p.	?	96.30%[4]	?	?
		WRN-28-10	−0.10 p.p.	?	77.78%[4]	?	?
			−0.27 p.p.	?	80.77%[4]	?	?
			−0.45 p.p.	?	86.30%[4]	?	?
	ImageNet	ResNet-18	+1.28 p.p.	?	65.75%[4]	?	?
			+1.64 p.p.	?	82.29%[4]	?	?
			−0.42 p.p.	?	91.14%[4]	?	?
			−7.94 p.p.	?	95.72%[4]	?	?
PR	CIFAR-10	DenseNet	−0.50 p.p.	?	74.37%	70.27%	?[5]
		WRN-28-10	+0.10 p.p.	?	76.94%	78.43%	75.00%
	ImageNet	MobileNetV2	+0.15 p.p.	?	90.00%	75.08%	74.23%
		ResNet-18	+0.14 p.p.	?	91.45%	80.06%	?[5]
EF	CIFAR-10	WRN-28-10	−0.60 p.p.	?	91.60%	92.10%	89.35%[6]
	ImageNet	ResNet-50	−1.00 p.p.	?	51.80%	53.20%	37.50%[6]
			−3.50 p.p.	?	73.00%	75.30%	66.89%[6]
			−7.70 p.p.	?	89.20%	89.90%	86.41%[6]

Table A.18.: Overview of the performance of works specifically targeting training and effect documented in original paper with comparable experiments missing in this thesis. Fields with % read as reduced by X%. LeNet-A=LeNet-300-100. [6]Unit of measurement no specified, FLOPs appears likely though within the context of the paper. [4]Not sparsity but tracked parameters during training (i.e., gradient sparsity). Abbreviations: EP = Eager Pruning, PT = Prunetrain, DP = Dropback, EF = Efficient Neural Network Training, PR = Procrustes, see also Table 3.6, FLOPS(T) = percentage by which FLOPs are decreased during training compared to a baseline training, FLOPS(I) = percentage by which FLOPs are decreased during inference compared to a baseline, C/F = convolution kernels or filters, P = parameters.

A.3. Additional Visualizations

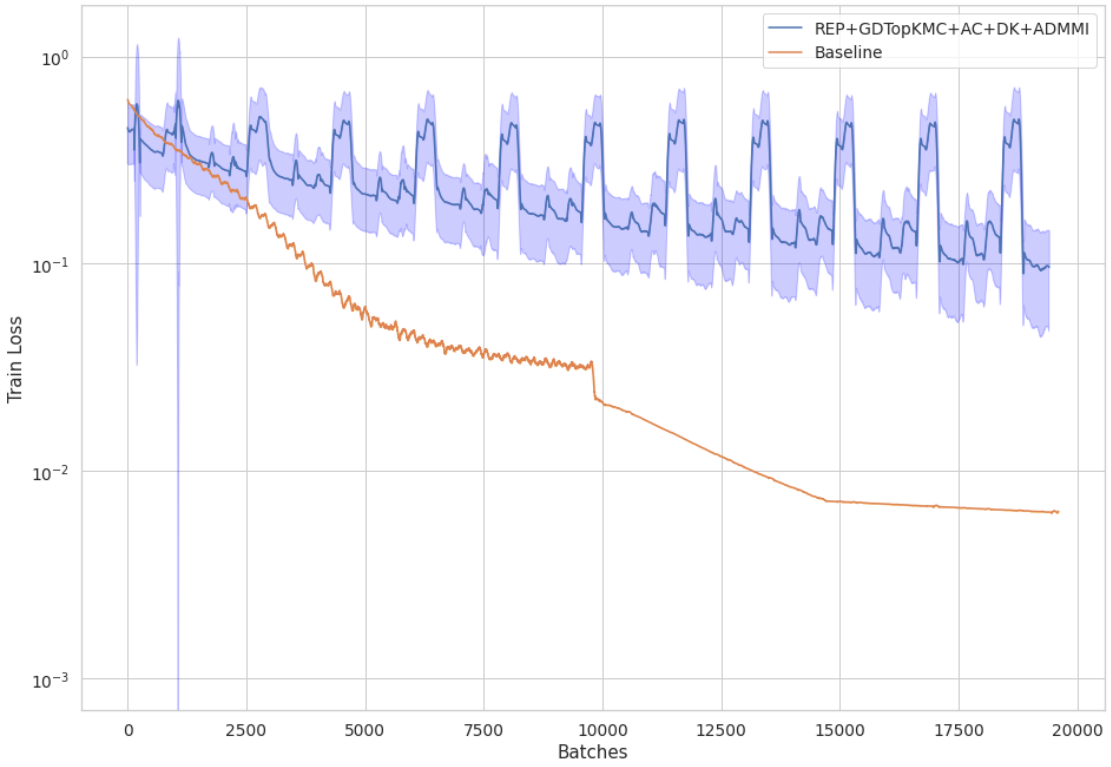


Figure A.1.: REP+GDTopKMC+AC+DK+ADMMI train loss on ResNet18/CIFAR-10 , 28 runs, mean and standard deviation, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability.

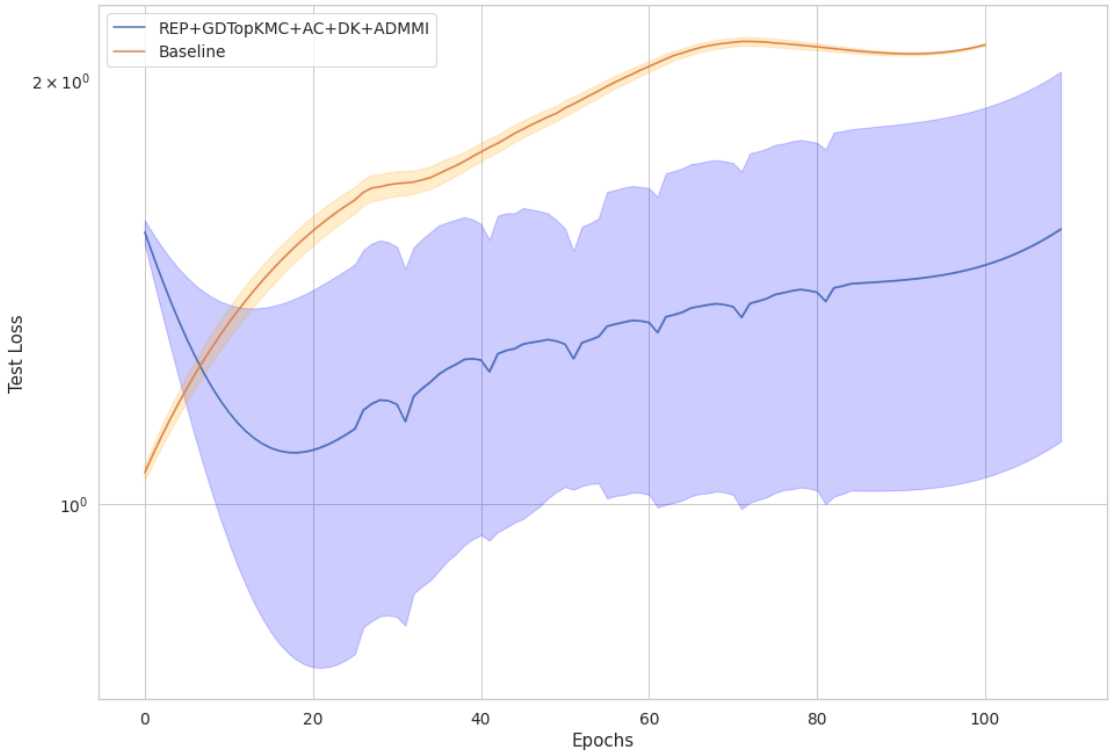


Figure A.2.: REP+GDTopKMC+AC+DK+ADMMI test loss on ResNet18/CIFAR-10, 28 runs, mean and standard deviation, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability.

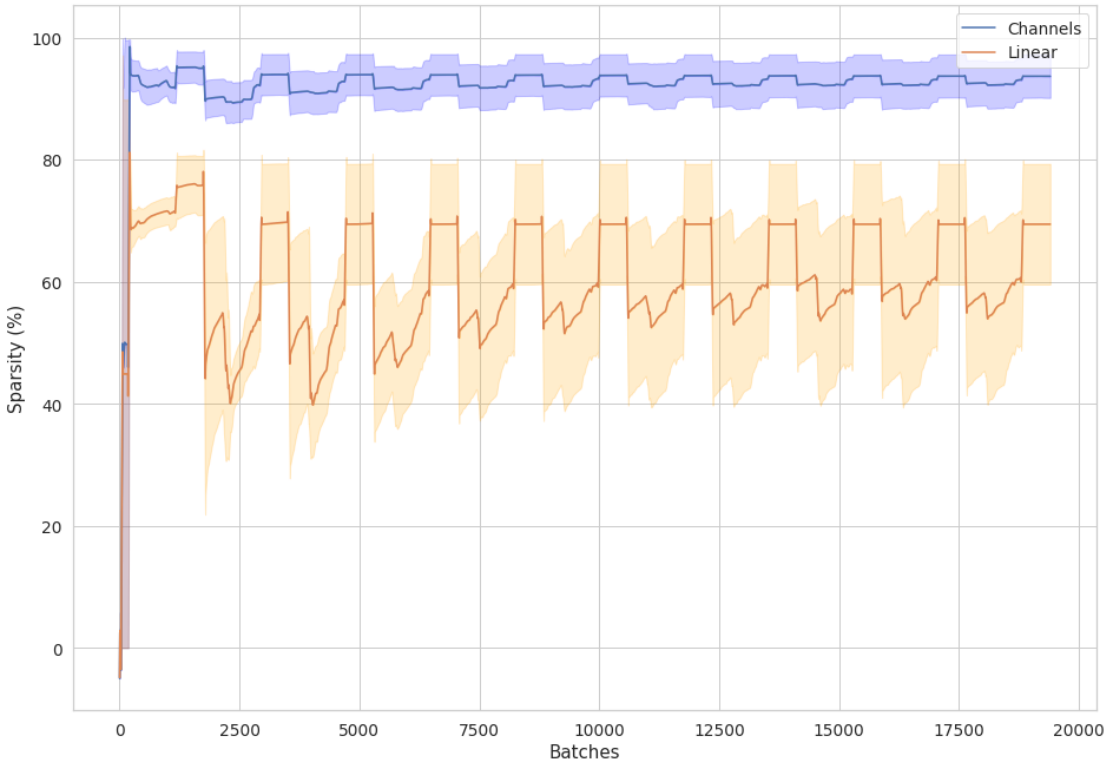


Figure A.3.: REP+GDTopKMC+AC+DK+ADMMI development of linear and channel sparsity during training of ResNet18/CIFAR-10, 28 runs, mean and standard deviation, smoothed with Savgol filter with window size 51 and polynomial of degree 3 to reduce jitter and improve visual evaluability.

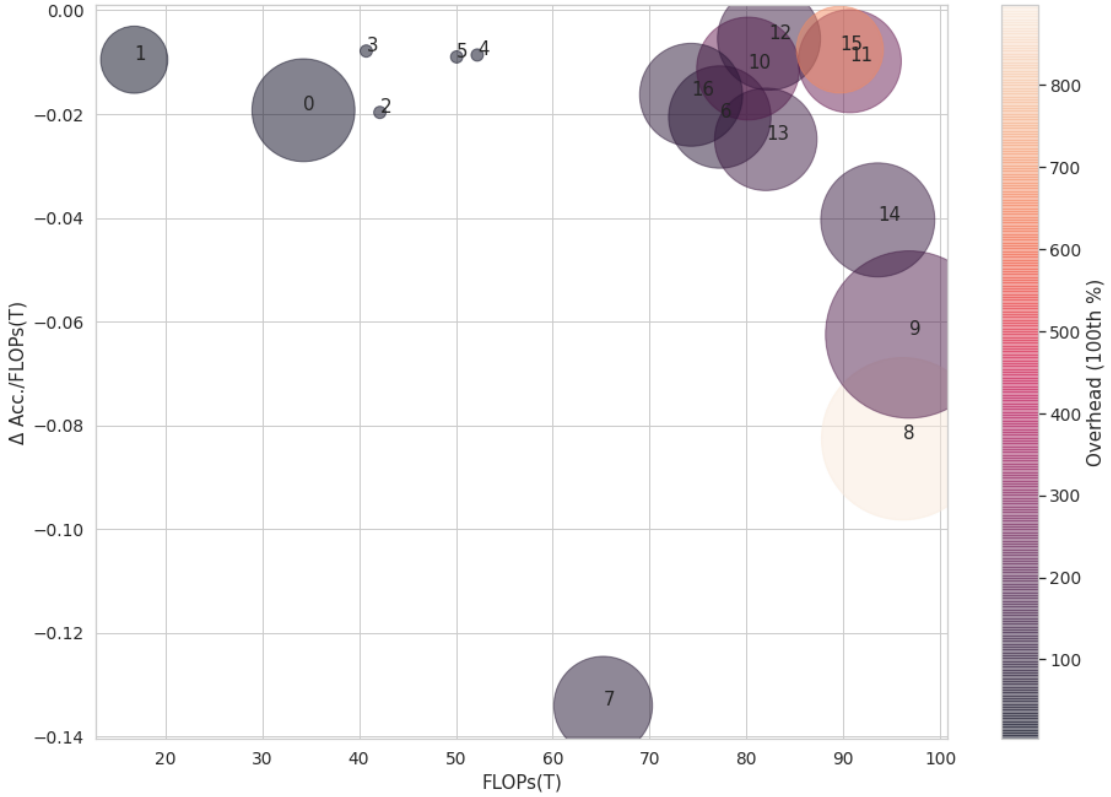


Figure A.4.: Ball scatter plot of results of Table 6.2 for LeNet-5/MNIST, 0=ADMMI, 1=ADMMR, 2=GDTopK, 3=GDTopKMC, 4=GDTopKMC+AC, 5=GDTopKMC+AC+DK, 6=GDTopKMC+AC+DK+ADMMI, 7=GDTopKMC+AC+DK+ADMMR, 8=REP, 9=REP+AC, 10=REP+AC+ADMMI, 11=REP+AC+ADMMR, 12=REP+ADMMI, 13=REP+ADMMR, 14=REP+GDTopKMC+AC+DK, 15=REP+GDTopKMC+AC+DK+ADMMI, 16=REP+GDTopKMC+AC+DK+ADMMR. Ball size encodes accuracy density (test accuracy per network density). Method overhead measured in hundredth % (denoted as 100th % in the figure) of total training FLOPs.

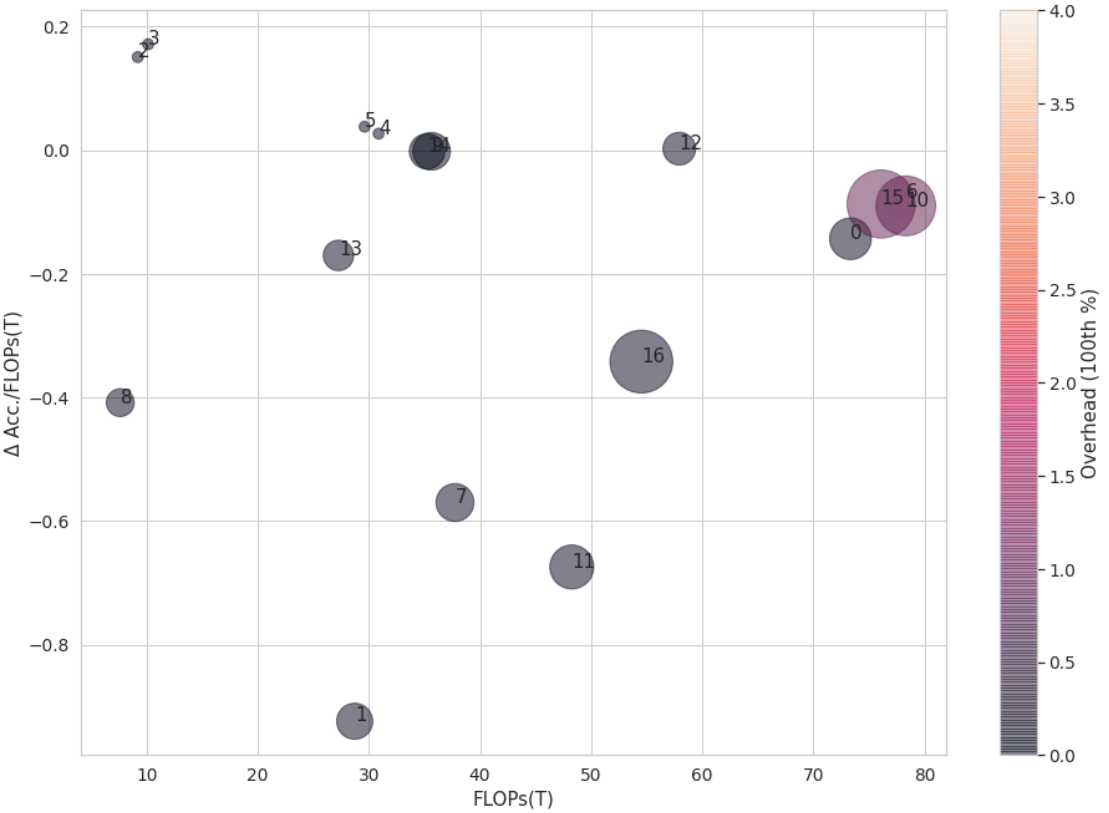


Figure A.5.: Ball scatter plot of results of Table 6.4 for ResNet18/CIFAR-10, 0=ADMMI, 1=ADMMR, 2=GDTopK, 3=GDTopKMC, 4=GDTopKMC+AC, 5=GDTopKMC+AC+DK, 6=GDTopKMC+AC+DK+ADMMI, 7=GDTopKMC+AC+DK+ADMMR, 8=REP, 9=REP+AC, 10=REP+AC+ADMMI, 11=REP+AC+ADMMR, 12=REP+ADMMI, 13=REP+ADMMR, 14=REP+GDTopKMC+AC+DK, 15=REP+GDTopKMC+AC+DK+ADMMI, 16=REP+GDTopKMC+AC+DK+ADMMR. Ball size encodes accuracy density (test accuracy per network density). Method overhead measured in hundredth % (denoted as 100th % in the figure) of total training FLOPs.

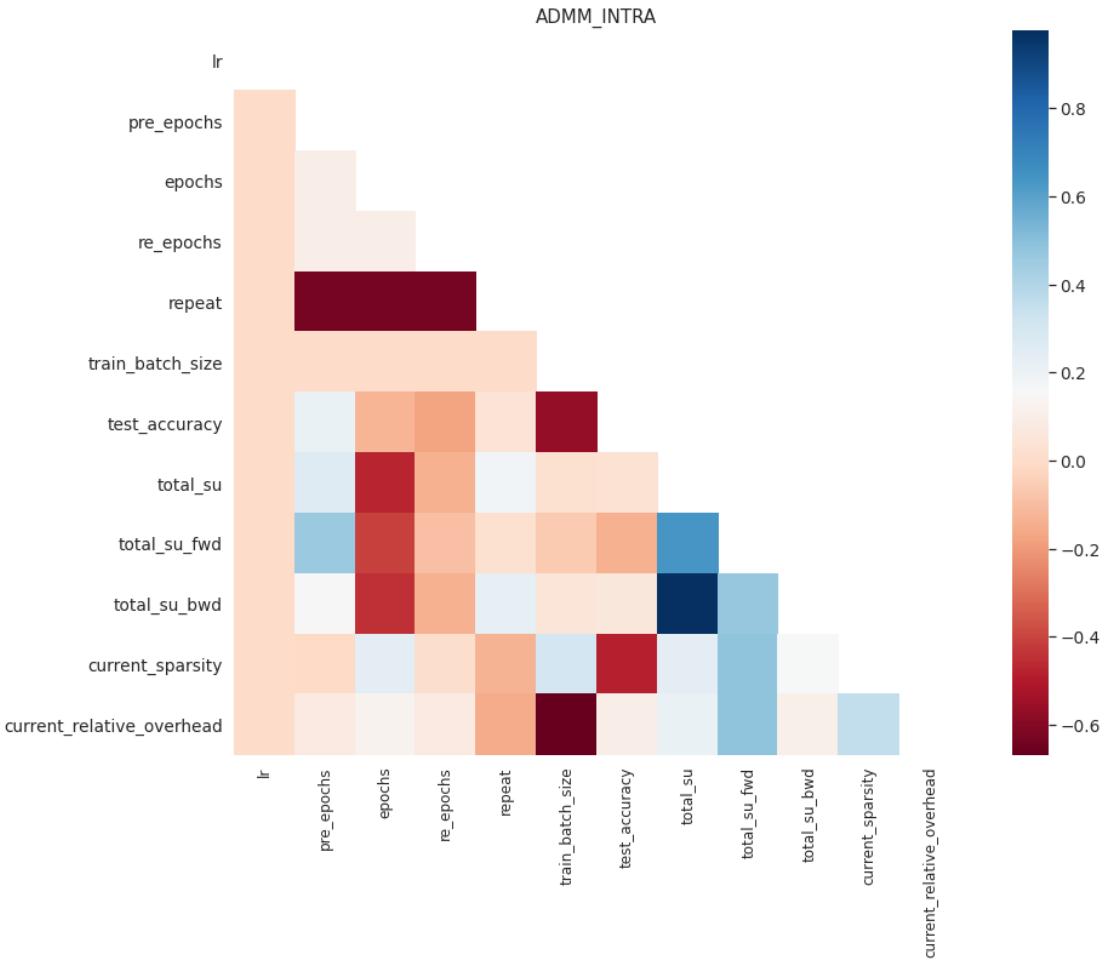


Figure A.6.: Correlation matrix for ADMMI. Please refer to Section 4.2 for details on individual parameters.

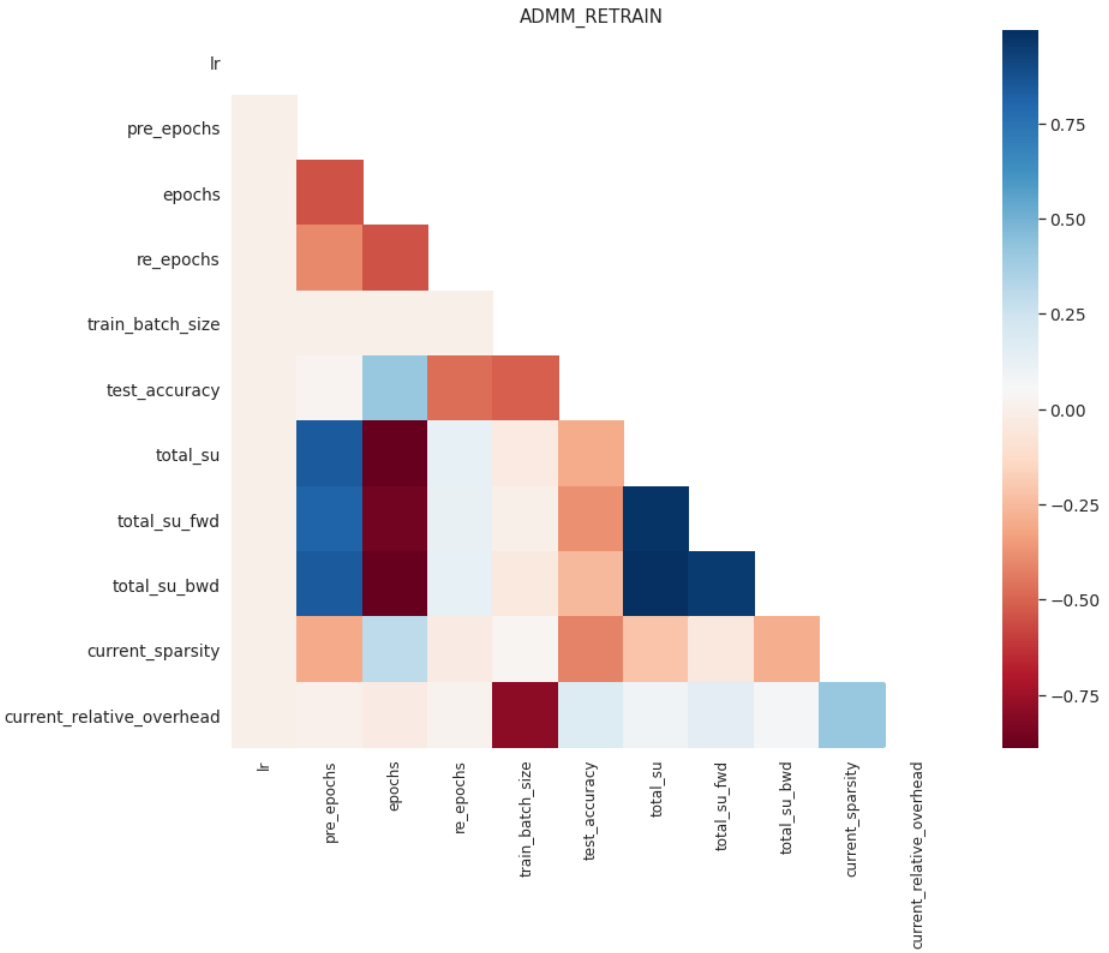


Figure A.7.: Correlation matrix for ADMMR. Please refer to Section 4.2 for details on individual parameters.

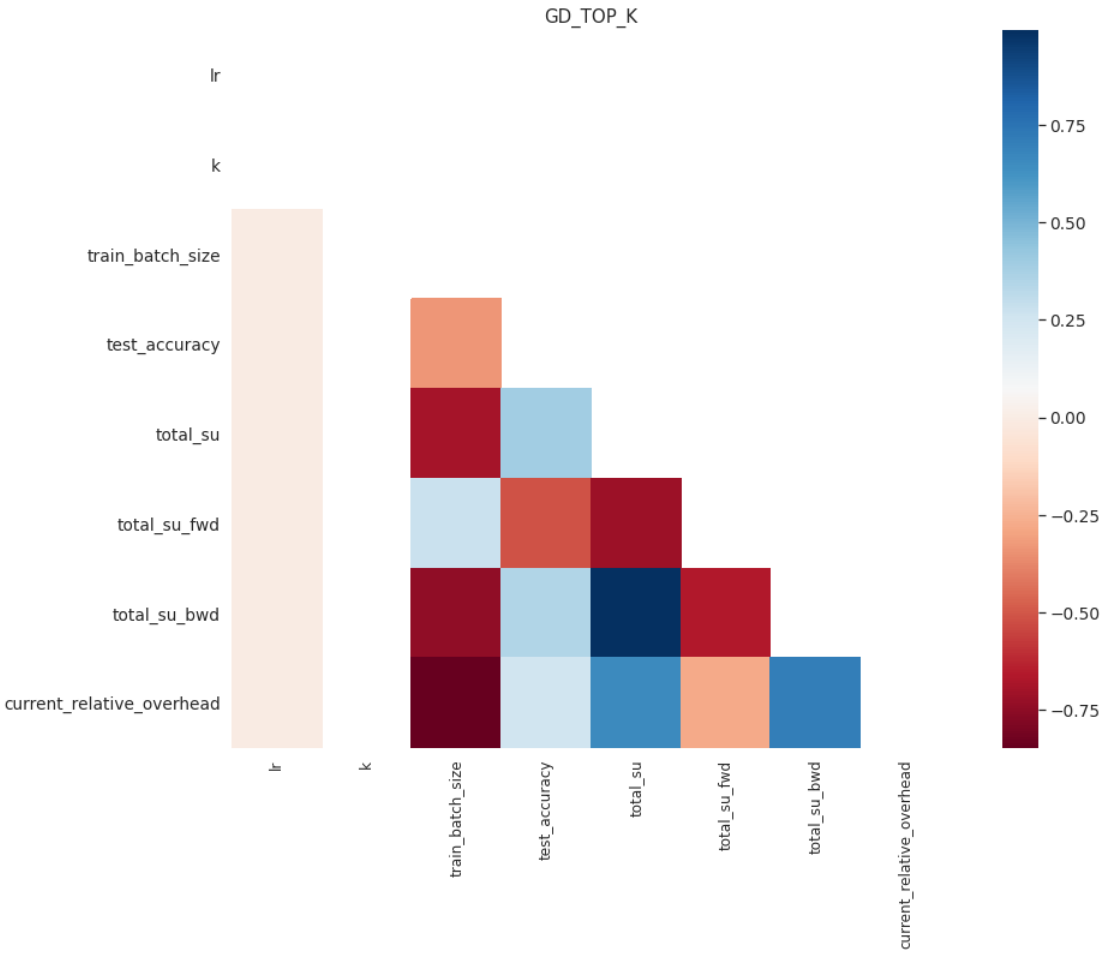


Figure A.8.: Correlation matrix for GDTopK. Please refer to Section 4.1.1.5 for details on individual parameters.

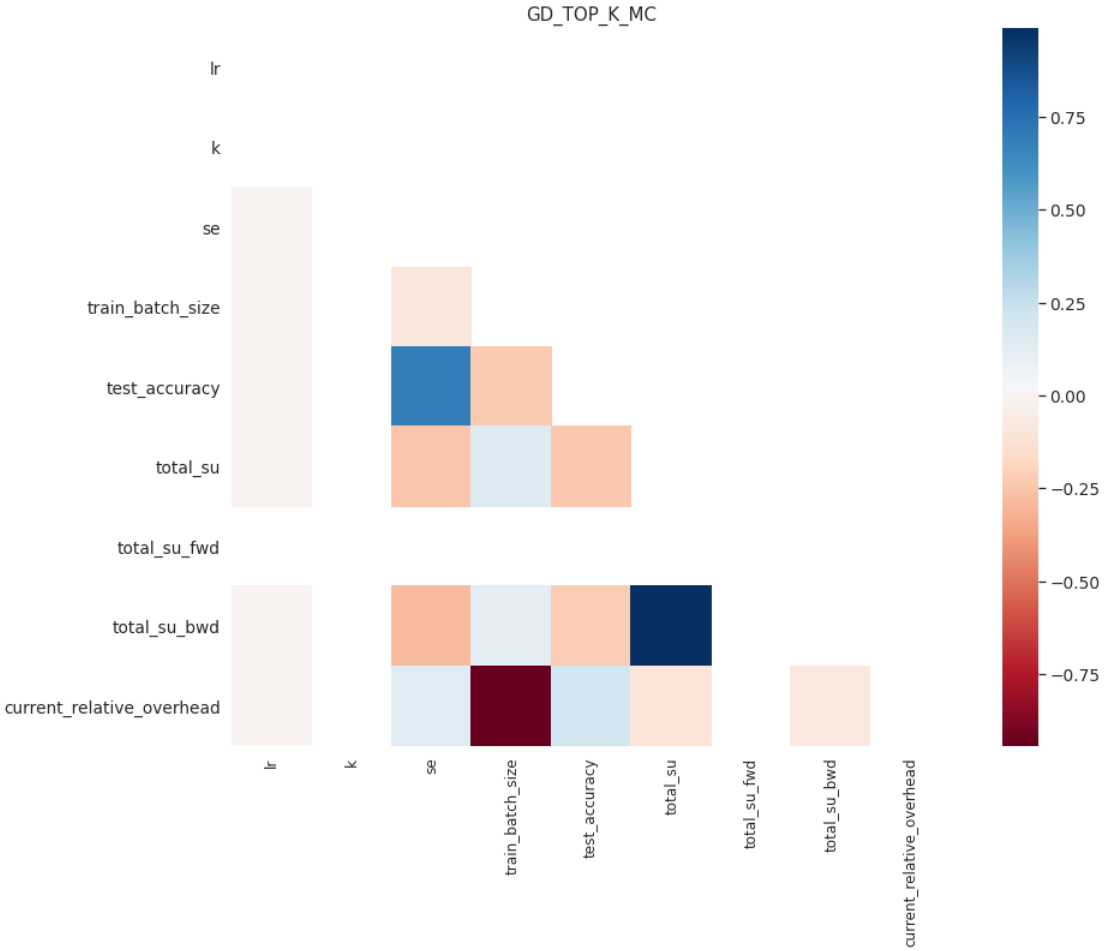


Figure A.9.: Correlation matrix for GDTopKMC. Please refer to Section 4.1.1.6 for details on individual parameters.

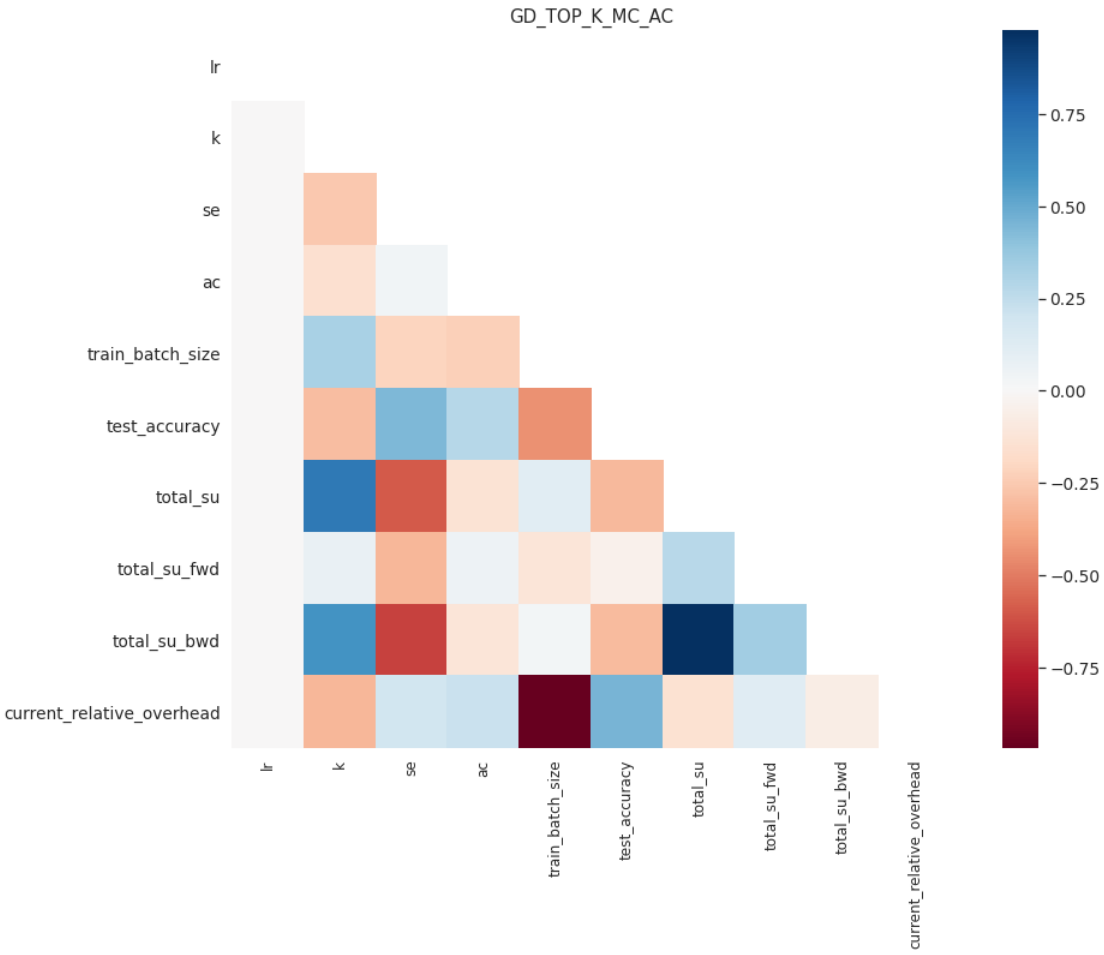


Figure A.10.: Correlation matrix for GDTopKMC+AC. Please refer to Sections 4.1.1.6 and 4.2 for details on individual parameters.

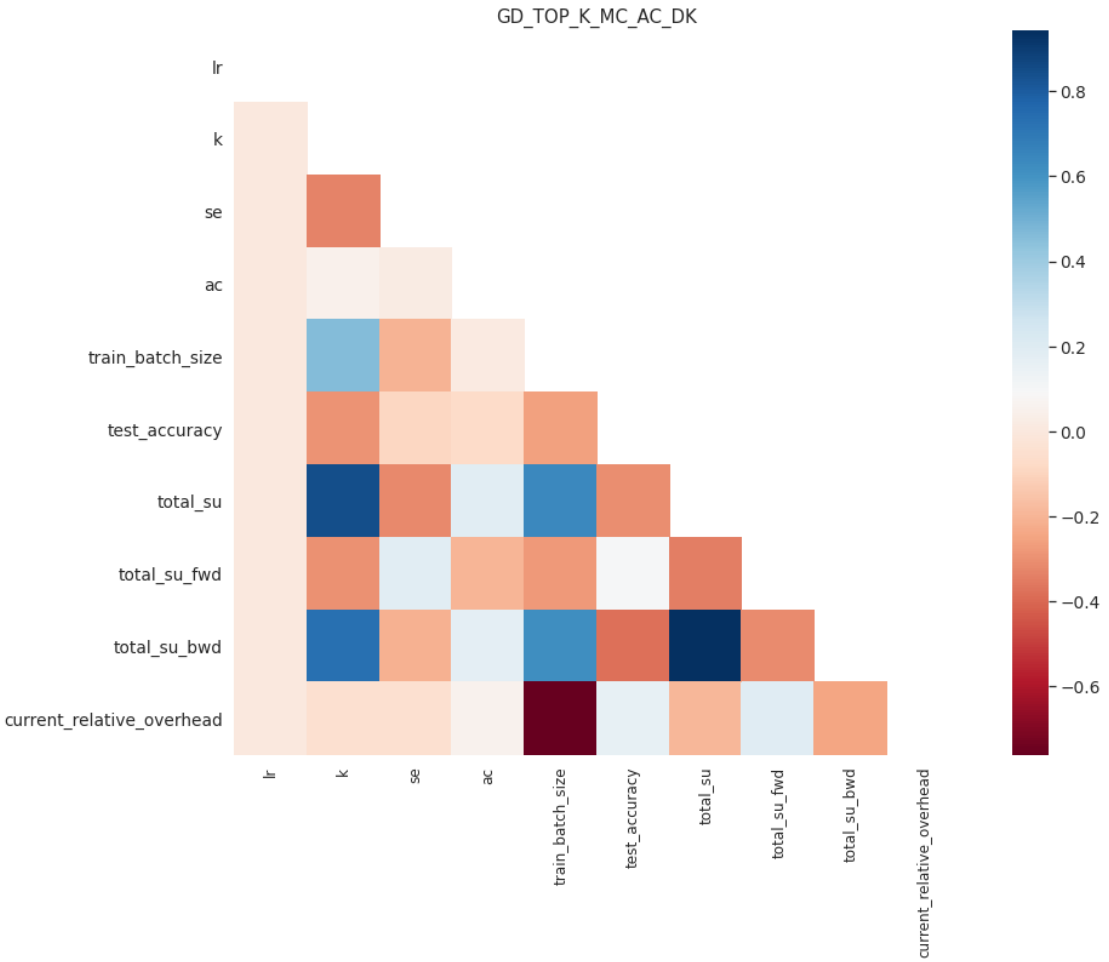


Figure A.11.: Correlation matrix for GDTopKMC+AC+DK. Please refer to Sections 4.1.1.6 and 4.2 for details on individual parameters.

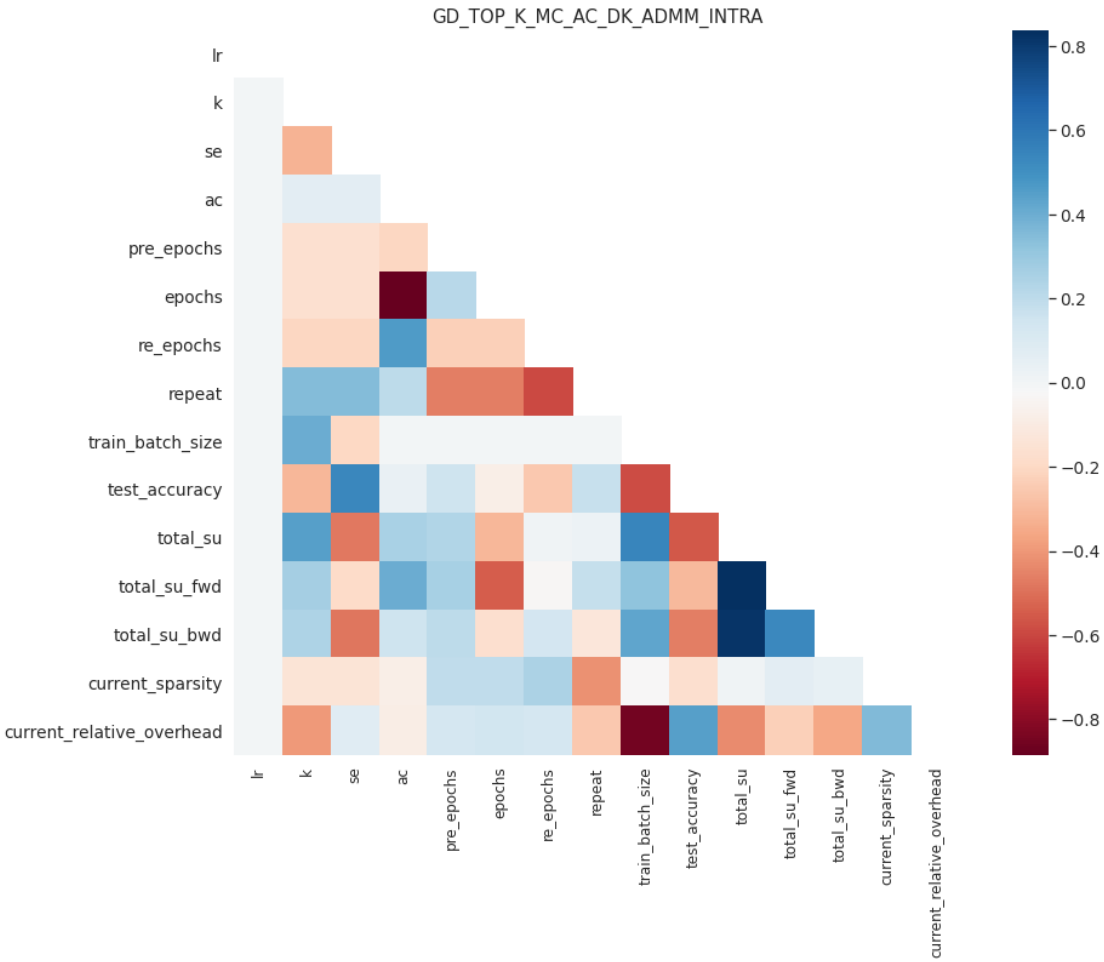


Figure A.12.: Correlation matrix for GDTopKMC+AC+DK+ADMMI. Please refer to Sections 4.1.1.6 and 4.2 for details on individual parameters.

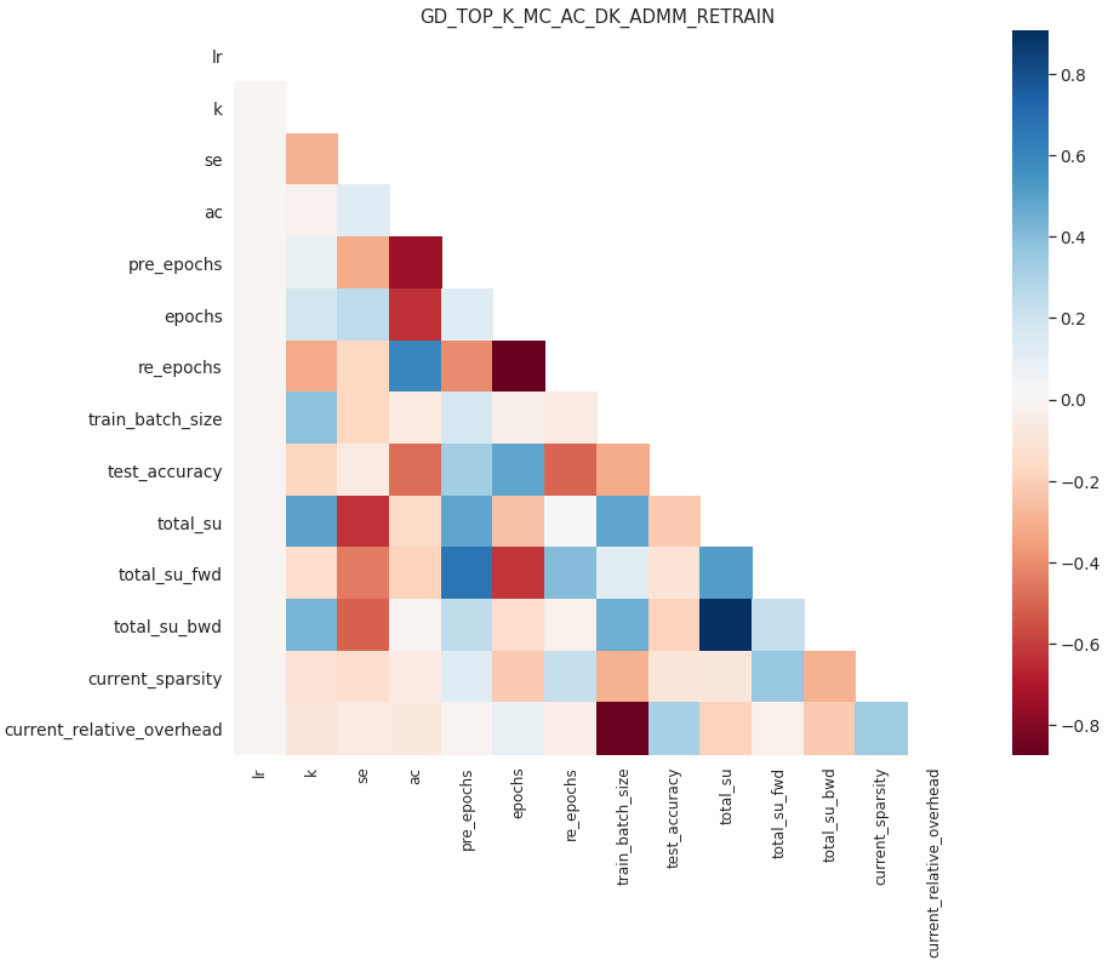


Figure A.13.: Correlation matrix for GDTopKMC+AC+DK+ADMMR. Please refer to Sections 4.1.1.6 and 4.2 for details on individual parameters.

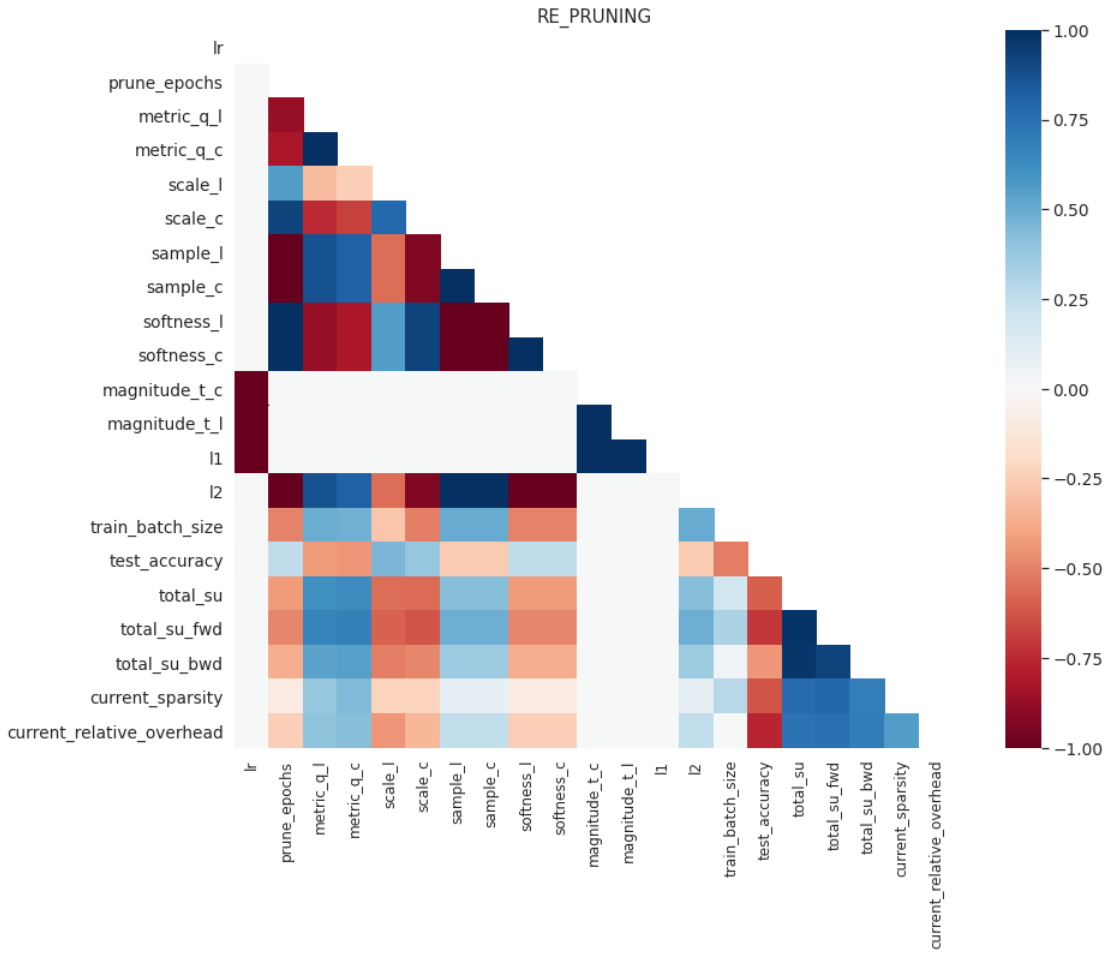


Figure A.14.: Correlation matrix for REP. Please refer to Section 4.1.2.4 for details on individual parameters.

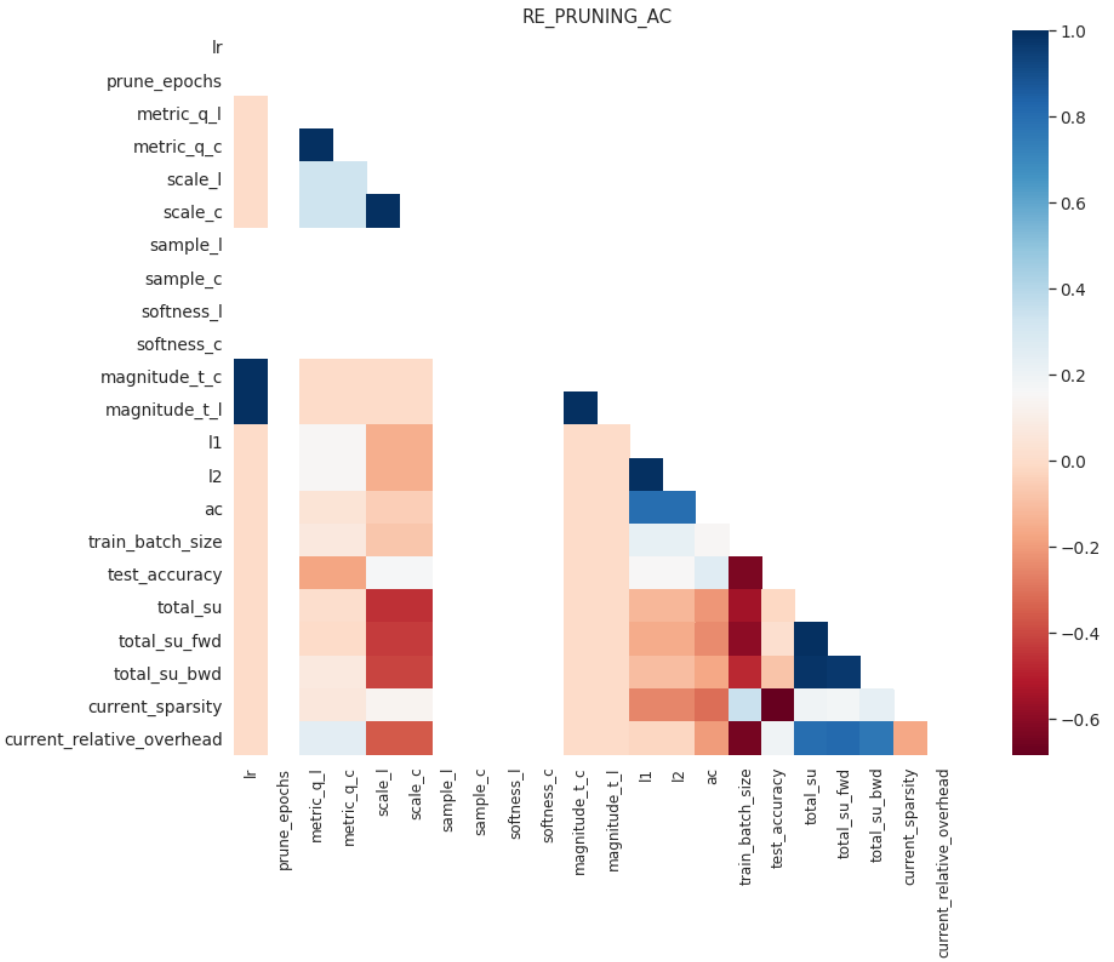


Figure A.15.: Correlation matrix for REP+AC. Please refer to Sections 4.1.2.4 and 4.2 for details on individual parameters.

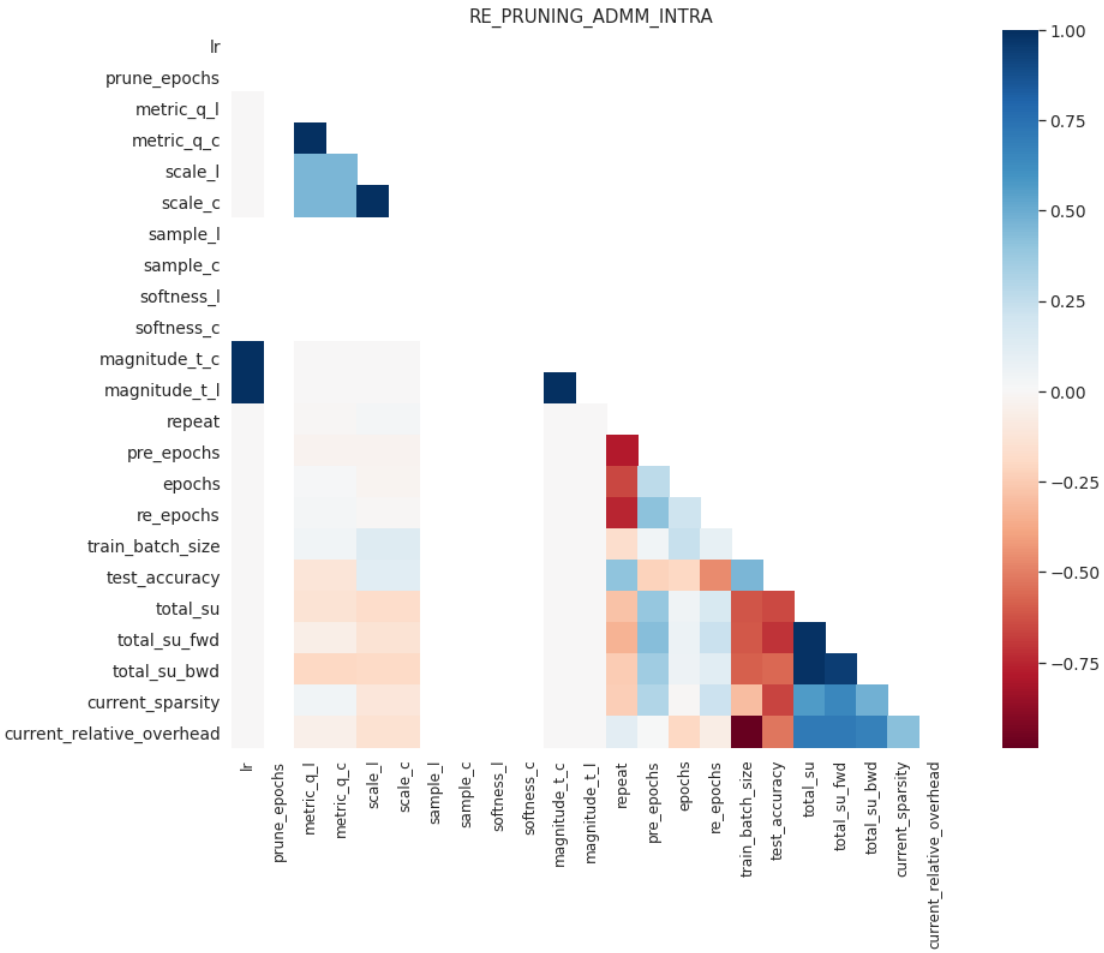


Figure A.16.: Correlation matrix for REP+ADMMI. Please refer to Sections 4.1.2.4 and 4.2 for details on individual parameters.

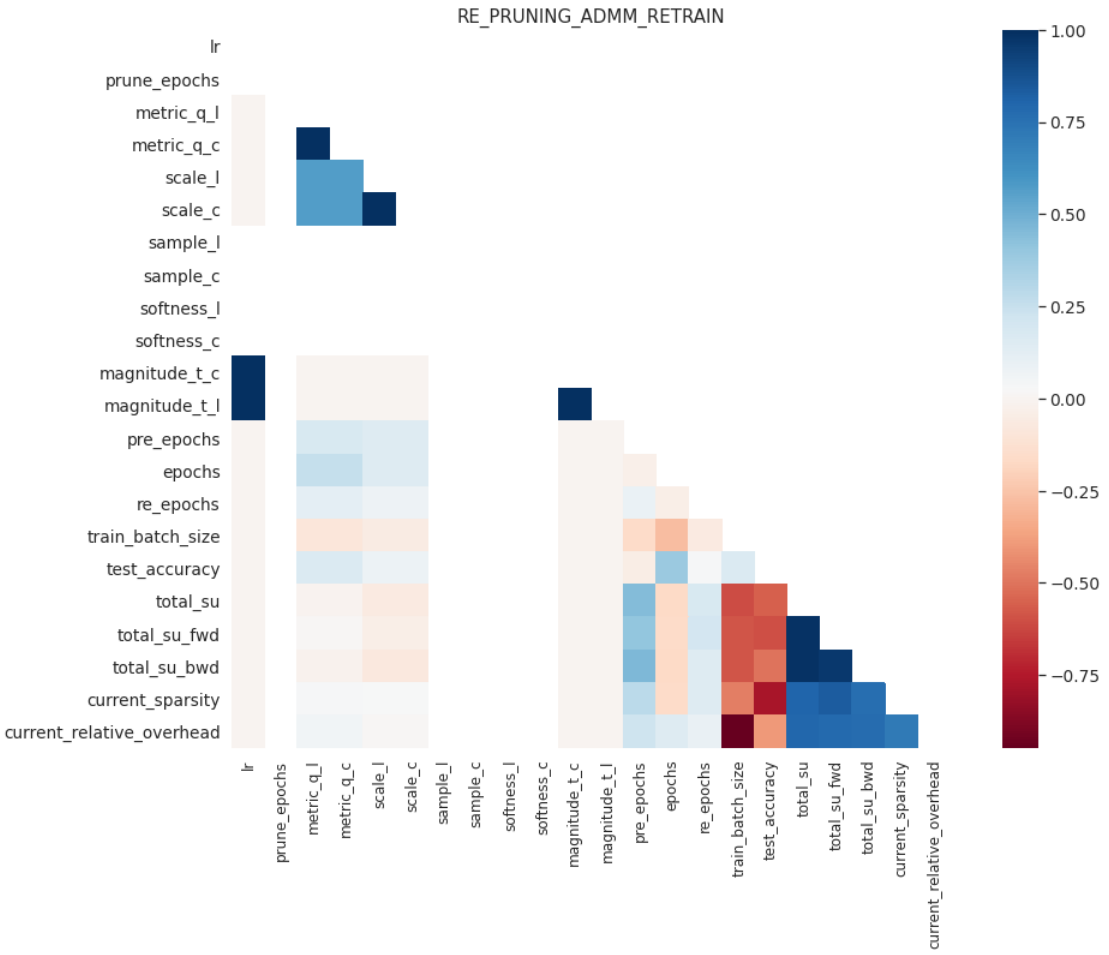


Figure A.17.: Correlation matrix for REP+ADMMR. Please refer to Sections 4.1.2.4 and 4.2 for details on individual parameters.

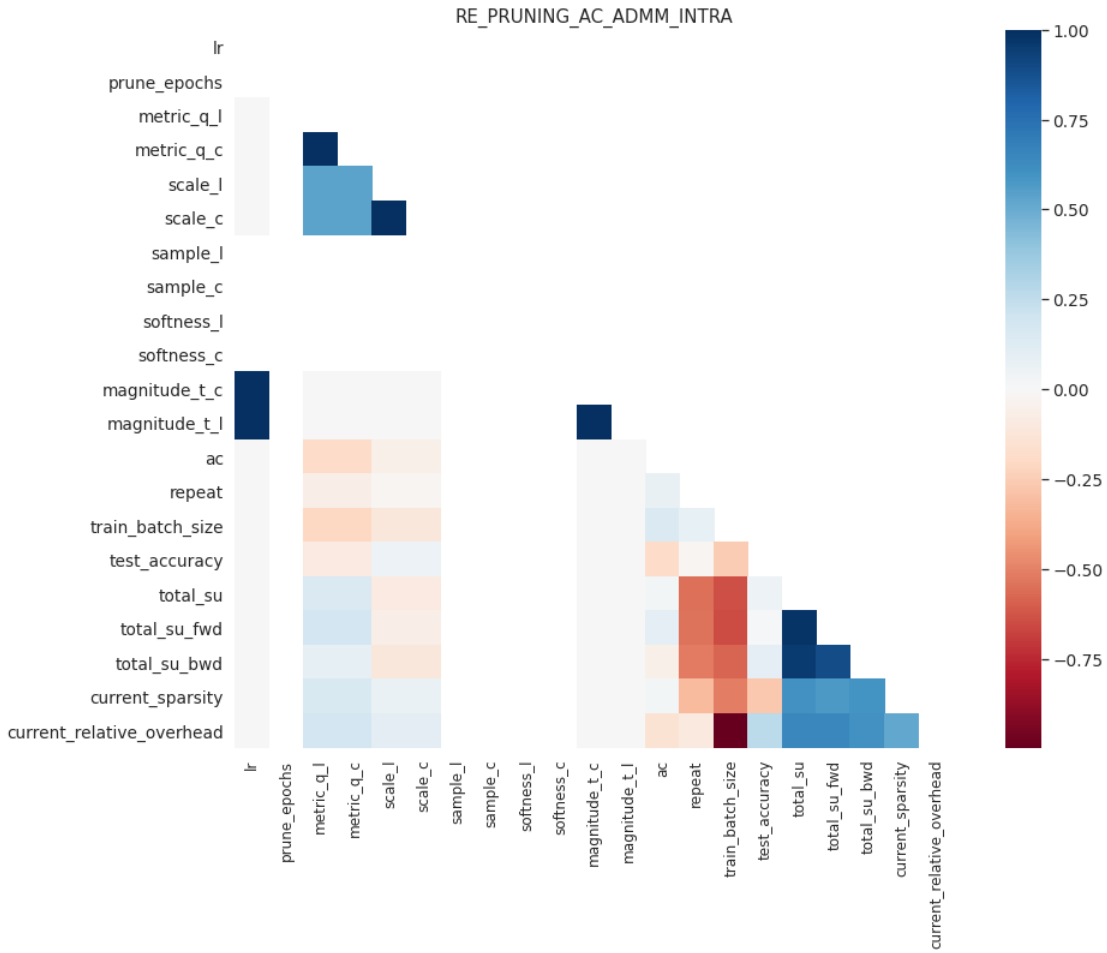


Figure A.18.: Correlation matrix for REP+AC+ADMMI. Please refer to Sections 4.1.2.4 and 4.2 for details on individual parameters.

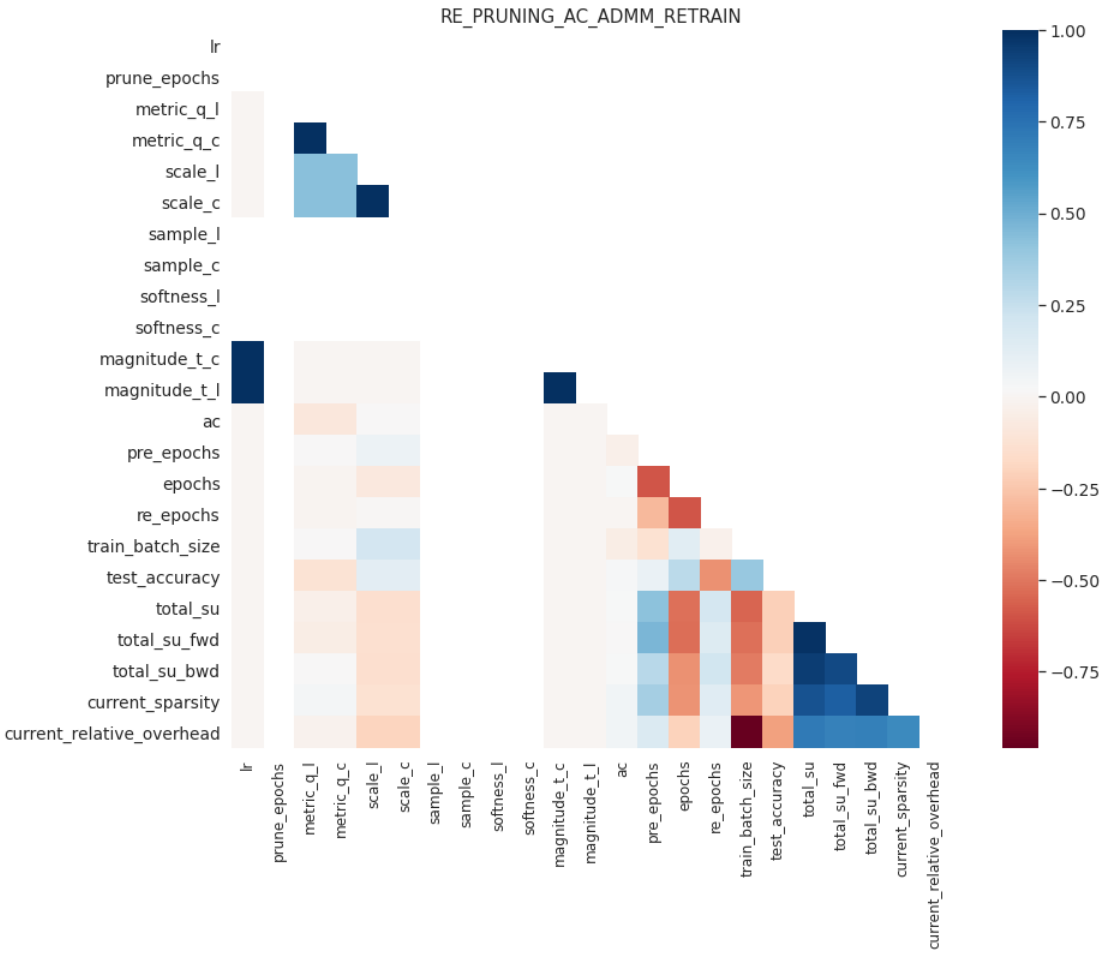


Figure A.19.: Correlation matrix for REP+AC+ADMMR. Please refer to Sections 4.1.2.4 and 4.2 for details on individual parameters.

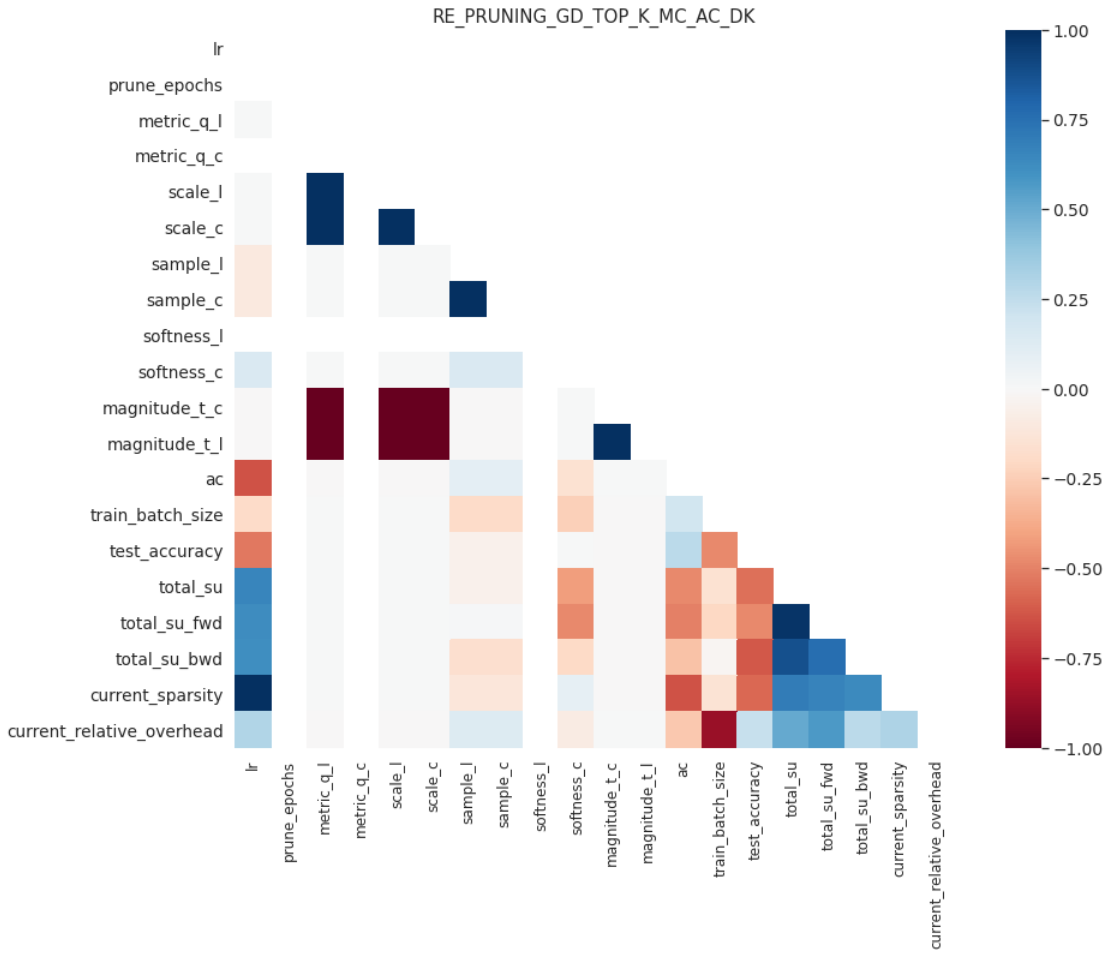


Figure A.20.: Correlation matrix for REP+GDTopKMC+AC+DK. Please refer to Sections 4.1.1.6, 4.1.2.4 and 4.2 for details on individual parameters.

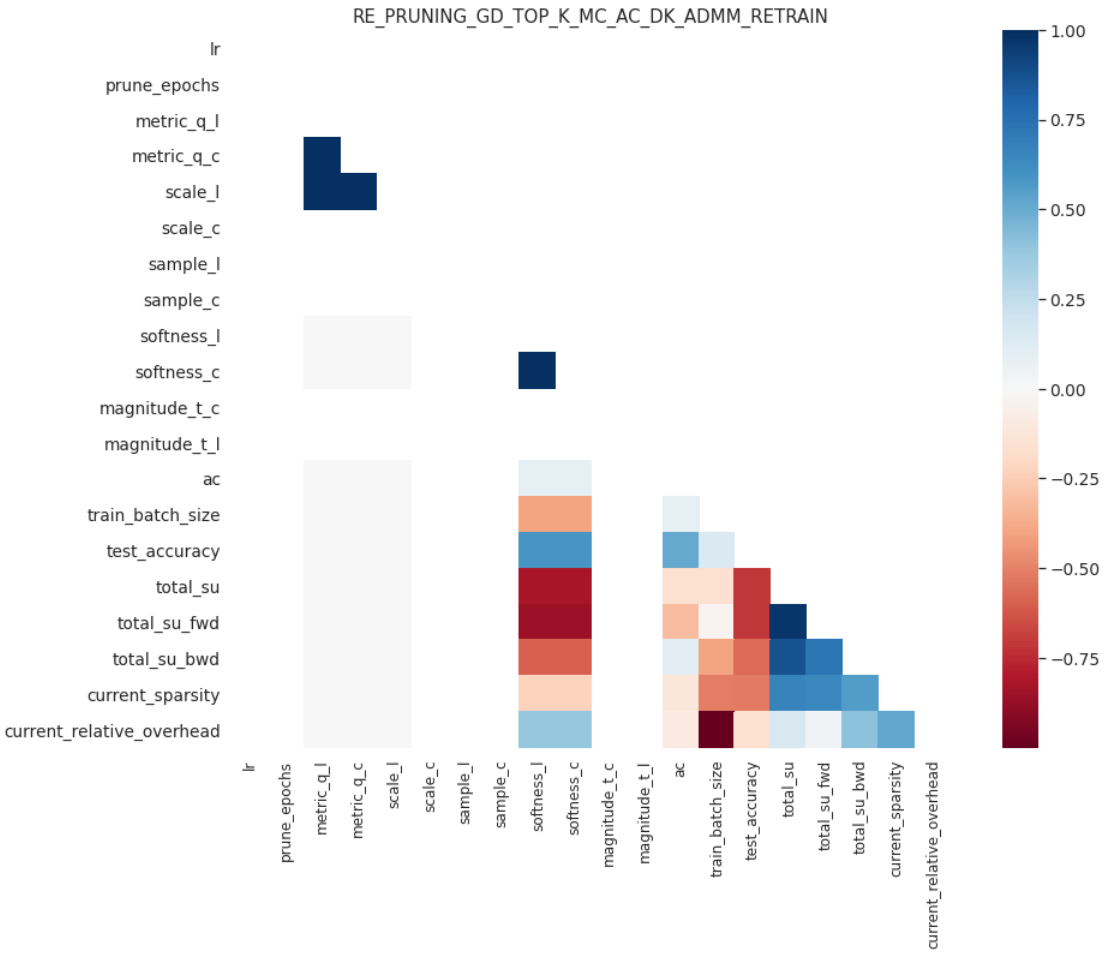


Figure A.21.: Correlation matrix for REP+GDTopKMC+AC+DK+ADMMR. Please refer to Sections 4.1.1.6, 4.1.2.4 and 4.2 for details on individual parameters.

Figure A.22.: REP+GDTopKMC+AC+DK+ADMMI induced sparsity structure in convolutional layer 1 of LeNet-5 trained on MNIST, black areas indicate zero elements, colored areas indicate non-zero elements/filters.

Figure A.23.: REP+GDTopKMC+AC+DK+ADMMI induced sparsity structure in linear layer 2 of LeNet-5 trained on MNIST, black areas indicate zero elements, colored areas indicate non-zero elements.