Resource-Efficient Neural Networks for Embedded Systems

Wolfgang Roth[*]   ROTH@TUGRAZ.AT
Graz University of Technology, Austria
Laboratory of Signal Processing and Speech Communication

Günther Schindler[*]   GUENTHER.SCHINDLER@ZITI.UNI-HEIDELBERG.DE
Heidelberg University, Germany
Institute of Computer Engineering

Bernhard Klein[*]   BERNHARD.KLEIN@ZITI.UNI-HEIDELBERG.DE
Heidelberg University, Germany
Institute of Computer Engineering

Robert Peharz   ROBERT.PEHARZ@TUGRAZ.AT
Graz University of Technology, Austria
Institute of Theoretical Computer Science

Sebastian Tschiatschek   SEBASTIAN.TSCHIATSCHEK@UNIVIE.AC.AT
University of Vienna, Austria
Faculty of Computer Science

Holger Fröning   HOLGER.FROENING@ZITI.UNI-HEIDELBERG.DE
Heidelberg University, Germany
Institute of Computer Engineering

Franz Pernkopf   PERNKOPF@TUGRAZ.AT
Graz University of Technology, Austria
Laboratory of Signal Processing and Speech Communication

Zoubin Ghahramani   ZOUBIN@ENG.CAM.AC.UK
University of Cambridge, UK

Editor: Russ Greiner

Abstract

While machine learning is traditionally a resource intensive task, embedded systems, autonomous navigation, and the vision of the Internet of Things fuel the interest in resource-efficient approaches. These approaches aim for a carefully chosen trade-off between performance and resource consumption in terms of computation and energy. The development of such approaches is among the major challenges in current machine learning research and key to ensure a smooth transition of machine learning technology from a scientific environment with virtually unlimited computing resources into everyday's applications. In this article, we provide an overview of the current state of the art of machine learning techniques facilitating these real-world requirements. In particular, we focus on resource-efficient inference based on deep neural networks (DNNs), the predominant machine learning models of the past decade. We give a comprehensive overview of the vast literature that can be mainly split into three non-mutually exclusive categories: (i) quantized neural networks, (ii) network pruning, and (iii) structural efficiency. These techniques can be applied dur-

---

[*] These authors contributed equally.

ing training or as post-processing, and they are widely used to reduce the computational demands in terms of memory footprint, inference speed, and energy efficiency. We also briefly discuss different concepts of embedded hardware for DNNs and their compatibility with machine learning techniques as well as potential for energy and latency reduction. We substantiate our discussion with experiments on well-known benchmark data sets using compression techniques (quantization, pruning) for a set of resource-constrained embedded systems, such as CPUs, GPUs and FPGAs. The obtained results highlight the difficulty of finding good trade-offs between resource efficiency and prediction quality.

Keywords: Resource-efficient machine learning, inference, deep neural networks.

1. Introduction

Machine learning is a key technology in the 21$^{\text{st}}$ century and the main contributing factor for many recent performance boosts in computer vision, natural language processing, speech recognition and signal processing. Today, the main application domain and comfort zone of machine learning applications is the "virtual world", as found in recommender systems, stock market prediction, and social media services. However, we are currently witnessing a transition of machine learning moving into "the wild", where most prominent examples are autonomous navigation for personal transport and delivery services, and the Internet of Things (IoT). Evidently, this trend opens several real-world challenges for machine learning engineers.
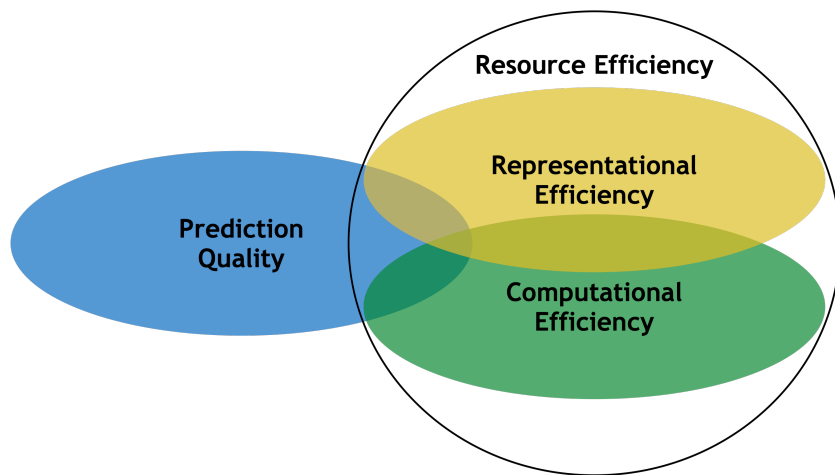

Figure 1: Aspects of resource-efficient machine learning models.

Current machine learning approaches prove particularly effective when large amounts of data and ample computing resources are available. However, in real-world applications the computing infrastructure during the operation phase is typically limited, which effectively rules out most of the current resource-hungry machine learning approaches. There are several key challenges—illustrated in Figure 1—which have to be jointly considered to facilitate machine learning in real-world applications:

Representational efficiency The model complexity in terms of memory footprint should match the (usually limited) resources in deployed systems. Model complexity is mainly governed by the employed model and its number of parameters. The selected numerical representations and sparsity in the parameters may also have an impact on the memory footprint and, therefore, contribute to the representational efficiency.

Computational efficiency The computational cost of performing inference should match the (usually limited) resources in deployed systems and exploit the available hardware optimally in terms of time and energy. Computational efficiency, in particular, also includes mapping the representational efficiency to available hardware structures. This is in contrast to theoretical inference costs, such as numbers of parameters and required mathematical operations, that often do not reflect inference running time on real hardware well. Furthermore, power constraints are key for autonomous and embedded systems, as the device lifetime for a given battery charge needs to be maximized, or constraints set by energy harvesters need to be met.

Prediction quality The focus of classical machine learning is mostly on optimizing the prediction quality of the models. For embedded devices, model complexity versus prediction quality trade-offs must be considered to achieve good prediction performance while simultaneously reducing computational complexity and memory requirements.

In this regard, resource-efficient neural networks for embedded systems are concerned with the trade-off between prediction quality and resource efficiency (i.e., representational efficiency and computational efficiency). This is highlighted in Figure 1. Note that this requires observing overall constraints such as prediction quality as well as inference latency and/or throughput, chip area and power consumption.

In this article, we review the state of the art in machine learning with regard to these real-world requirements. We focus on deep neural networks (DNNs), the currently predominant machine learning models. We formally define DNNs in Section 2 and give a brief introduction to the most prominent building blocks, such as dropout and batch normalization. While being the driving factor behind many recent success stories, DNNs are notoriously data and resource hungry, a property which has recently renewed significant research interest in resource-efficient approaches. This paper is dedicated to giving an extensive overview of the current directions of research of these approaches, all of which are concerned with reducing the model size and/or improving inference efficiency while at the same time maintaining accuracy levels close to state-of-the-art models. We have identified three major directions of research concerned with enhancing resource efficiency in DNNs that we present in Section 3. In particular, these directions are:

Quantized Neural Networks Typically, the weights of a DNN are stored as 32-bit floating-point values and during inference millions of floating-point operations are carried out. Quantization approaches reduce the number of bits used to store the weights and the activations of DNNs. While quantization approaches obviously reduce the memory footprint of a DNN, the selected weight representation potentially also facilitates faster inference using cheaper arithmetic operations. Even reducing precision down to binary or ternary values works reasonably well and essentially reduces DNNs to hardware-friendly logical circuits.

Network Pruning Starting from a fixed, potentially large DNN architecture, pruning approaches remove parts of the architecture during training or after training as a post-processing step. The parts being removed range from the very local scale of individual weights—which is called unstructured pruning—to a more global scale of neurons, channels, or even entire layers—which is called structured pruning. On the one hand, unstructured pruning is typically less sensitive to accuracy degradation, but special sparse matrix operations are required to obtain a computational benefit. On the other hand, structured pruning is more delicate with respect to accuracy but the resulting data structures remain dense such that common highly optimized dense matrix operations, available on most off-the-shelf hardware, can be used.

Structural Efficiency This category comprises a diverse set of approaches that achieve resource efficiency at the structural level of DNNs. *Knowledge distillation* is an approach where a small student DNN is trained to mimic the behavior of a larger teacher DNN, which has been shown to yield improved results compared to training the small DNN directly. The idea of *weight sharing* is to use a small set of weights that is shared among several connections of a DNN to reduce the memory footprint. Several works have investigated *special matrix structures* that require fewer parameters and allow for faster matrix multiplications—the main workload in fully connected layers. Furthermore, there exist several *manually designed architectures* that introduced lightweight building blocks or modified existing building blocks to enhance resource efficiency. Most recently, *neural architecture search* methods have emerged that discover efficient DNN architectures automatically.

Evidently, many of the presented techniques are not mutually exclusive and can potentially be combined to further enhance resource efficiency. For instance, one can both sparsify a model and reduce arithmetic precision.

We complement our literature review with a brief overview of embedded hardware for DNNs in Section 4. These hardware platforms can be categorized into CPUs, GPUs, FPGAs and domain-specific accelerators, where each architecture exhibits different properties for deploying models. We discuss potentials and limitations of such embedded hardware with considerations on vectorization and parallelization, frequency and energy efficiency, as well as their applicability for resource-efficient models.

In Section 5 we substantiate our discussion with experimental results. We provide a comparison of various quantization approaches for DNNs using the CIFAR-100 data set in Section 5.1.1, followed by an evaluation of prediction quality for different types of pruned structures on the CIFAR-10 data set in Section 5.1.2. We evaluate the inference throughput of the compressed models on an ARM CPU (Section 5.2.1), Xilinx FPGA (Section 5.2.2) and an embedded NVIDIA GPU (Section 5.2.3). We conclude the experiments with an overall comparison in Section 5.2.4, where the embedded systems (in combination with compression techniques) are studied with respect to inference throughput and prediction quality.

2. Background

Before we present a comprehensive overview of the many different techniques for reducing the complexity of DNNs in Section 3, this section formally introduces DNNs and some fundamentals required in the remainder of the paper.

2.1 Feed-forward Deep Neural Networks

DNNs are typically organized in layers of alternating linear transformations and non-linear activation functions. A vanilla DNN with L layers is a function mapping an input $\mathbf{x}^0$ to an output $y = \mathbf{x}^L$ by applying the iterative computation

$$\mathbf{a}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l, \tag{1}$$

$$\mathbf{x}^l = \phi(\mathbf{a}^l), \tag{2}$$

where (1) computes a linear transformation with weight tensor $\mathbf{W}^l$ and bias vector $\mathbf{b}^l$, and (2) computes a non-linear activation function ϕ that is typically applied element-wise. Common choices for ϕ are the ReLU function $\phi(a) = \max(a, 0)$, sigmoid functions, such as $\tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$ and the logistic function $1/(1 + e^{-a})$, and, in the context of resource-efficient models, the sign function $\text{sign}(a) = \mathbb{I}(a \geq 0) - \mathbb{I}(a < 0)$, where $\mathbb{I}$ is the indicator function.

In this paper, we focus on hardware-efficient machine learning in the context of classification, i.e., the task of assigning the input $\mathbf{x}^0$ to a class $\hat{c} \in \{1, \ldots, \mathcal{C}\}$. Other predictive tasks, such as regression and multi-label prediction, can be tackled in a similar manner. For classification tasks, the output activation function ϕ for computing $\mathbf{x}^L \in \mathbb{R}^{\mathcal{C}}$ is typically the softmax function $\phi(\mathbf{a})_i = e^{a_i} / \sum_j e^{a_j}$. An input $\mathbf{x}^0$ is assigned to class $\hat{c} = \arg\max_c x_c^L$.

The two most common types of layers are (i) fully connected layers[1] and (ii) convolutional layers. For fully connected layers, the input $\mathbf{x} \in \mathbb{R}^n$ is a vector whose individual dimensions—also called *neurons*[2]—do not exhibit any a-priori known structure. The linear transformation of a fully connected layer is implemented as a matrix-vector multiplication $\mathbf{W}\mathbf{x}$ where $\mathbf{W} \in \mathbb{R}^{m \times n}$.

Convolutions are used if the data exhibits spatial or temporal dimensions such as images, in which case the DNN is called a convolutional neural network (CNN). Two-dimensional images can be represented as three-dimensional tensors $\mathbf{x}^l \in \mathbb{R}^{C \times W \times H}$, where C refers to the number of *channels* (or, equivalently, *feature maps*), and W and H refer to the width and the height of the image, respectively. A $K_w \times K_h$ convolution using a rank-4 *filter* weight tensor $\mathbf{W} \in \mathbb{R}^{K_w \times K_h \times C \times D}$ mapping $\mathbf{x}^l \in \mathbb{R}^{C \times W \times H}$ to $\mathbf{a}^{l+1} \in \mathbb{R}^{D \times W \times H}$ is computed as

$$a_{d,w,h}^{l+1} = \sum_{k_w=1}^{K_w} \sum_{k_h=1}^{K_h} \sum_{c=1}^{C} W_{k_w,k_h,c,d} \cdot x_{c,i(w,k_w,K_w),i(h,k_h,K_h)}^l, \tag{3}$$

where i is the auxiliary indexing function

$$i(p, k, K) = p - \left\lceil \frac{K}{2} \right\rceil + k. \tag{4}$$

---

1. Many popular deep learning frameworks refer to fully connected layers as *dense* layers.
2. For $1 < l < L$, we speak of *hidden* layers and *hidden* neurons.

Each spatial location of the output feature map $\mathbf{a}^{l+1}$ is computed from a $K_w \times K_h$ region of the input image $\mathbf{x}^l$. By using the same filter to compute the values at different spatial locations, a translation invariant detection of features is obtained. The spatial size of features detected within an image is bounded by the *receptive field*, i.e., the section of the input image that influences the value of a particular spatial location in some hidden layer. The receptive field is increased by stacking multiple convolutional layers, e.g., performing two consecutive 3×3 convolutions results in each output spatial location being influenced by a larger 5×5 region of the input feature maps.

Another form of translational invariance is achieved by *pooling* operations that merge spatially neighboring values within a feature map to reduce the feature map's size. Common choices are max-pooling and average-pooling which combine the results of neighboring values[3] by computing their maximum or average, respectively. Furthermore, pooling operations also increase the receptive field.

2.2 Training of Deep Neural Networks

The task of training is concerned with adjusting the weights $\mathbf{W}$ such that the DNN reliably predicts correct classes for unseen inputs $\mathbf{x}^0$. This is accomplished by minimizing a loss function $\mathcal{L}$ using gradient-based optimization (Nocedal and Wright, 2006). Given some labeled training data $\mathcal{D} = \{(\mathbf{x}_1^0, t_1), \ldots, (\mathbf{x}_N^0, t_N)\}$ containing N input-target pairs, a typical loss function has the form

$$\mathcal{L}(\mathbf{W}; \mathcal{D}) = \sum_{n=1}^{N} l(y(\mathbf{W}, \mathbf{x}_n^0), t_n) + \lambda r(\mathbf{W}), \tag{5}$$

where $l(y_n, t_n)$ is the *data term* that penalizes the DNN parameters $\mathbf{W}$ if the output y_n does not match the target value t_n, $r(\mathbf{W})$ is a *regularizer* that prevents the DNN from overfitting, and $\lambda > 0$ is a trade-off hyperparameter. Typical choices for the data term $l(y_n, t_n)$ are the cross-entropy loss or the mean squared error loss, whereas typical choices for the regularizer $r(\mathbf{W})$ are the ℓ^1-norm or the ℓ^2-norm of the weights. The loss is minimized using *gradient descent* by iteratively computing

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}; \mathcal{D}), \tag{6}$$

where η is a learning rate hyperparameter. In practice, more involved *stochastic* gradient descent (SGD) schemes, such as ADAM (Kingma and Ba, 2015), are used that randomly select smaller subsets of the data—called mini-batches—to approximate the gradient.

Modern deep learning frameworks play an important role in the growing popularity of DNNs as they make gradient-based optimization particularly convenient: The user specifies the loss $\mathcal{L}$ as a computation graph and the gradient $\nabla_{\mathbf{W}} \mathcal{L}$ is calculated automatically by the framework using the backpropagation algorithm (Rumelhart et al., 1986).

2.3 Batch Normalization

The literature has established a consensus that using more layers improves the classification performance of DNNs. However, increasing the number of layers L also increases the difficulty of training a DNN using gradient-based methods as described in Section 2.2. Most

---

3. Typically, a 2×2 region to halve the feature map size is used.

modern DNN architecture employ batch normalization (Ioffe and Szegedy, 2015) after the linear transformation of some or all layers by computing

$$a_{n,d}^l \leftarrow \frac{a_{n,d}^l - \mu_d^l}{\sigma_d^l} \cdot \gamma_d + \beta_d \quad \text{with} \quad \mu_d^l \leftarrow \frac{1}{N_B} \sum_{n=1}^{N_B} a_{n,d}^l, \quad (\sigma_d^l)^2 \leftarrow \frac{1}{N_B - 1} \sum_{n=1}^{N_B} (a_{n,d}^l - \mu_d^l)^2,$$
(7)

where β_d and γ_d are trainable parameters, and N_B is the mini-batch size of SGD.

The idea is to normalize the activation statistics over the data samples in each layer to zero mean and unit variance. This results in similar activation statistics throughout the network which facilitates gradient flow during backpropagation. The linear transformation of the normalized activations with the parameters β and γ is mainly used to recover the DNNs ability to approximate any desired function—a feature that would be lost if only the normalization step is performed. Most recent DNN architectures have been shown to benefit from batch normalization, and, as reviewed in Section 3.2.2, batch normalization can be targeted to achieve resource efficiency in DNNs.

2.4 Dropout

Dropout as introduced by Srivastava et al. (2014) is a way to prevent neural networks from overfitting by injecting multiplicative noise to the inputs of a layer, i.e., $x_d^l \leftarrow x_d^l \cdot \varepsilon$. A common choice for the injected noise is $\varepsilon \sim \text{Bernoulli}(p)$ where the values x_d^l are randomly set to zero with probability $1 - p$. Another prevalent option is Gaussian dropout, where we sample ε from a normal distribution with mean 1 and variance α.[4] Intuitively, the idea is that hidden neurons cannot rely on the presence of features computed by other neurons. Consequently, individual neurons are expected to compute in a sense "meaningful" features on their own. This avoids multiple neurons jointly computing features in an entangled way. Dropout has been cast into a Bayesian framework which was subsequently exploited to perform network pruning as detailed in Section 3.2.3.

2.5 Common Neural Architectures

As stated earlier in this section, the majority of architectures follow the simple pattern of repeating several layers of linear transformation followed by a non-linear function ϕ. Although most successful architectures follow this scheme, recent architectures have introduced additional components and subtle extensions that have led to new design principles. In the following, we give a brief overview of the most prominent architectures that have emerged over the past years in chronological order.

2.5.1 ALEXNET

The AlexNet architecture (Krizhevsky et al., 2012) was the first work to show that DNNs are capable of improving performance over conventional hand crafted computer vision techniques by achieving 16.4% Top-5 error on the ILSVRC12 challenge—an improvement of approximately 10% absolute error compared to the second best approach in the challenge

---

4. The parameters p and α are hyperparameters.

which relied on well-established computer vision techniques. This most influential work essentially started the advent of DNNs, which have since spread to virtually all scientific domains, often achieving improvements over well-established methods in their respective fields.

The architecture consists of eight layers—five convolutional layers followed by three fully connected layers. AlexNet was designed to optimally utilize the available hardware at that time rather than following some clear design principle. This involves the choice of heterogeneous window sizes $K_w \times K_h$ and seemingly arbitrary numbers of channels per layer C. Furthermore, convolutions are performed in two parallel paths (i.e., grouped convolutions; see Section 3.3.4) to facilitate the training on two GPUs.

2.5.2 VGGNet

The VGGNet architecture (Simonyan and Zisserman, 2015) won the second place at the ILSVRC14 challenge with 7.3% Top-5 error. Compared to AlexNet, its structure is more uniform and with up to 19 layers much deeper. The design of VGGNet is guided by two main principles. (i) VGGNet uses mostly 3×3 convolutions and increases the receptive field by stacking several of them. (ii) After downscaling the spatial dimension with 2×2 max-pooling, the number of channels should be doubled to avoid information loss. From a hardware perspective, VGGNet is often preferred over other architectures due to its uniform architecture.

2.5.3 InceptionNet

InceptionNet (or, equivalently, GoogLeNet) (Szegedy et al., 2015) won the ILSVRC14 challenge with 6.7% Top-5 error with an even deeper architecture consisting of 22 layers. The main feature of this architecture is the inception module which combines the outputs of 1×1, 3×3, and 5×5 convolutions by stacking them. To reduce the computational burden, InceptionNet performs 1×1 convolutions as proposed in (Lin et al., 2014a) to reduce the number of channels immediately before the larger 3×3 and 5×5 convolutions.

2.5.4 ResNet

Motivated by the observation that adding more layers to very deep conventional CNN architectures does not necessarily reduce the *training* error, residual networks (ResNets) introduced by He et al. (2016) follow a rather different principle. The key idea is that every layer computes a residual that is *added* to the layer's input. This is often graphically depicted as a residual path with an attached skip connection.

The authors hypothesize that identity mappings play an important role. They argue that it is easier to model identity mappings in ResNets by simply setting all the weights of the residual path to zero instead of simulating them by adapting the weights of several consecutive layers in an intertwined way. In any case, the skip connections reduce the vanishing gradient problem during training and enable extremely deep architectures of up to 152 layers on ImageNet and even up to 1,000 layers on CIFAR-10. ResNet won the ILSVRC15 challenge with 3.6% Top-5 error.

2.5.5 DENSENET

Inspired by ResNets whose skip connections have shown to reduce the vanishing gradient problem, densely connected CNNs (DenseNets) introduced by Huang et al. (2017) drive this idea even further by connecting each layer to *all* previous layers. DenseNets are conceptually very similar to ResNets—instead of adding the output of a layer to its input, DenseNets *stack* the output and the input of each layer. Since this stacking necessarily increases the number of feature maps with each layer, the number of new feature maps computed by each layer is typically small. Furthermore, it is proposed to use compression layers after downscaling the spatial dimension with pooling, i.e., a 1×1 convolution is used to reduce the number of feature maps.

Compared to ResNets, DenseNets achieve similar performance, allow for even deeper architectures, and they are more parameter and computation efficient. However, the DenseNet architecture is highly non-uniform which complicates the hardware mapping and ultimately slows down training.

2.5.6 MOBILENET

The core idea of MobileNet (Howard et al., 2017b) is to improve resource efficiency by depthwise-separable convolutions, which decompose an expensive standard convolution into two cheaper sequential convolutions. In the first convolution, a filter is applied to each channel separately without taking other channels into account. The second convolution restores information flow across channels by performing a 1×1 convolution. By using depthwise-separable convolutions, the number of trainable parameters as well as the number of multiply-accumulate operations (MACs) can be substantially reduced. It is empirically shown that this has little to no negative impact on prediction quality. MobileNetV2 (Sandler et al., 2018a) extends this concept by introducing additive skip connections and bottleneck layers. MobileNetV3 (Howard et al., 2019) extends this even further by also incorporating the neural architecture search (NAS) proposed in MnasNet (Tan et al., 2018). More details will be provided in Section 3.3.4 and Section 3.3.5.

2.5.7 EFFICIENTNET

EfficientNet (Tan and Le, 2019, 2021) combines a selection of previous works on resource efficiency with regard to skip connections, depthwise-separable convolutions, Squeeze-and-Excitation (SE) modules, and NAS.

The intuition of Squeeze-and-Excitation networks (SENet) (Iandola et al., 2016) is to learn channel importances which is relevant as the number of channels typically becomes larger with an increasing depth of the architecture. A squeeze module first reduces information by fusing information from each feature map into a single value, e.g., by global average pooling. This results in a vector of length C, which encodes a feature descriptor of the whole feature map. This is followed by an excitation module for adaptive recalibration that captures channel-wise dependencies by learning.

NAS is used to find a baseline architecture called EfficientNet-B0. This model is similar to MnasNet which is composed of multiple stages where each stage is based on one or more layers of depthwise-separable convolutions and SE modules. While EfficientNet-B0 is the smallest model, enlarged variants of this base model are created by using *compound scaling.*

These upscaled variants are identical to the base model in terms of number of stages and stage type, but for each stage the input feature map resolution, width and depth is scaled.

2.5.8 TRANSFORMERS

While all previously presented architectures are based on convolutional architectures, recently transformers have gained significant attention in natural language processing (NLP), speech processing and computer vision, representing a deviation from traditional sequence processing architectures like recurrent and convolutional neural architectures. The core of the transformer architecture (Vaswani et al., 2017) is the self-attention mechanism, which allows to weight the importance of different parts of the input sequence. This mechanism enables transformers to capture long-range dependencies in data, which can be challenging for recurrent and convolutional architectures. Furthermore, unlike convolutional layers, which have fixed receptive fields, transformers can handle input sequences of variable lengths, making them more flexible and suitable for many tasks. While transformers were initially developed for sequence-to-sequence tasks in NLP, they have also been explored as replacements for convolutional neural networks in computer vision tasks.

One notable example is the Vision Transformer (ViT) (Dosovitskiy et al., 2021), where the input image is first divided into non-overlapping patches of fixed size. Each patch is then linearly embedded into a flat vector, forming the initial input to the transformer. To understand the spatial relationships between patches (position of each patch in the image), a positional encoding is added to these embeddings. The patch embeddings in combination with this positional encoding is then processed by the transformer encoder, such that the self-attention mechanism captures dependencies between different patches and enables the understanding of the global context of the image. In combination with class tokens, the classification head can then produce classification scores.

While this self-attention mechanism is extremely successful for various tasks, it comes at substantial processing complexity. A self-attention operation involves computing attention scores for each pair of positions in the input sequence, resulting in a complexity of N^2 for a sequence of length N. Similarly, the corresponding attention matrix has a size of $N \times N$. As transformers typically use multiple attention heads in parallel to capture different aspects of the relationships between input elements, this complexity is further scaled by the number of attention heads H.

Sparse attention mechanisms and approximations have been proposed to address this issue and improve the efficiency of transformers for longer sequences. We refer to the work of Tay et al. (2022) which provides an overview of various transformer-based architectures that focus on efficiency, reduced memory-footprint and computational complexity. Most of these methods focus on the quadratic complexity of the self-attention heads and use low-rank matrix operations, downsampling or exploit pre-set or learned sparsity patterns.

2.6 The Straight-Through Gradient Estimator

Many recently developed methods for resource efficiency in DNNs incorporate components in the computation graph of the loss function $\mathcal{L}$ that are non-differentiable or whose gradient is zero almost everywhere, such as piecewise constant quantizers. These components prevent the use of conventional gradient-based optimization as described in Section 2.2.
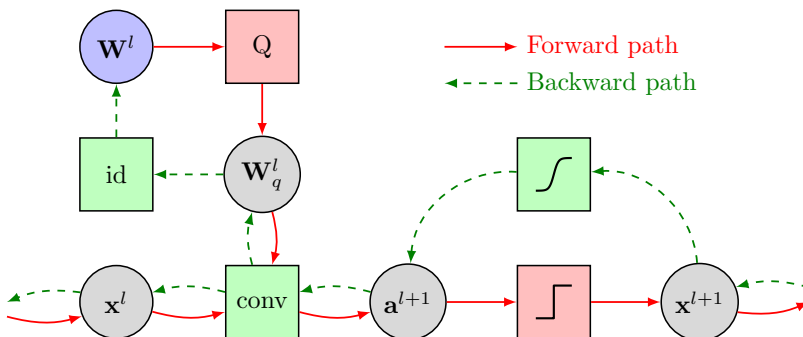
Figure 2: A simplified building block of a DNN using the straight-through gradient estimator (STE). Q denotes some arbitrary piecewise constant quantization function and id denotes the identity function which simply passes the gradient on during backpropagation. In the forward pass, the solid red line is followed which passes the two piecewise constant functions Q and sign whose gradient is zero almost everywhere (red boxes). During backpropagation, the dashed green line is followed which avoids these piecewise constant functions and instead only passes differentiable functions (green boxes)—in particular, the functions id and tanh whose shapes are similar to Q and sign but whose gradient is non-zero. This allows us to obtain an approximate non-zero gradient for the real-valued parameters $\mathbf{W}^l$ (blue circle) which are subsequently updated with SGD.

The straight-through gradient estimator (STE) is a simple yet effective way to approximate the gradient of such components by simply replacing their gradient with a non-zero value. Let $f(w)$ be some non-differentiable operation within the computation graph of $\mathcal{L}$ such that the partial derivative $\partial\mathcal{L}/\partial w$ is not defined. The STE then approximates the gradient $\partial\mathcal{L}/\partial w$ by

$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\partial\mathcal{L}}{\partial f}\frac{\partial f}{\partial w} \approx \frac{\partial\mathcal{L}}{\partial f}\frac{\partial\tilde{f}}{\partial w}, \tag{8}$$

where $\tilde{f}(w)$ is an arbitrary differentiable function with a similar functional shape as $f(w)$. For instance, in case of the sign activation function $f(w) = \text{sign}(w)$ whose derivative is zero almost everywhere, one could select $\tilde{f}(w) = \tanh(w)$. Another common choice is the identity function $\tilde{f}(w) = w$ whose derivative is $\tilde{f}'(w) = 1$, which simply passes the gradient on to higher components in the computation graph during backpropagation. Figure 2 illustrates the STE applied to a simplified DNN layer.

2.7 Bayesian Neural Networks

Since there exist several works for resource-efficient DNNs that build on the framework of Bayesian neural networks, we briefly introduce the basic principles here. Given a prior distribution $p(\mathbf{W})$ over the weights and a likelihood $p(\mathcal{D}|\mathbf{W})$ defined by the softmax output

11

of a DNN as

$$p(\mathcal{D}|\mathbf{W}) = \prod_{n=1}^{N} p(y(\mathbf{W}, \mathbf{x}_n^0) = t_n), \tag{9}$$

we can use Bayes' rule to infer a posterior distribution over the weights, i.e.,

$$p(\mathbf{W}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{W})\, p(\mathbf{W})}{p(\mathcal{D})} \propto p(\mathcal{D}|\mathbf{W})\, p(\mathbf{W}). \tag{10}$$

From a Bayesian perspective it is desired to compute expected predictions with respect to the posterior distribution, i.e.,

$$\mathbb{E}_{p(\mathbf{W}|\mathcal{D})}[y(\mathbf{W}, \mathbf{x}^0)], \tag{11}$$

and not just to reduce the entire distribution to a single point estimate. However, due to the highly non-linear nature of DNNs, most exact inference scenarios involving the full posterior $p(\mathbf{W}|\mathcal{D})$ are typically intractable and there exist a range of approximation techniques for these tasks, such as variational inference (Hinton and van Camp, 1993; Graves, 2011; Blundell et al., 2015) and sampling based approaches (Neal, 1992).

Interestingly, training DNNs can often be seen as a very rough Bayesian approximation where we only seek for weights $\mathbf{W}$ that maximize the posterior $p(\mathbf{W}|\mathcal{D})$, which is also known as maximum a-posteriori estimation (MAP). In particular, in a typical loss $\mathcal{L}$ as in (5) the data term originates from the logarithm of the likelihood $p(\mathcal{D}|\mathbf{W})$ whereas the regularizer originates from the logarithm of the prior $p(\mathbf{W})$.

A better Bayesian approximation is obtained with variational inference where the aim is to find a variational distribution $q(\mathbf{W}|\nu)$ governed by distribution parameters ν that is as close as possible to the posterior $p(\mathbf{W}|\mathcal{D})$ but still simple enough to allow for efficient inference, e.g., for computing $\mathbb{E}_{q(\mathbf{W}|\nu)}[y(\mathbf{W}, \mathbf{x}^0)]$ by sampling from q. This is typically achieved by the so called *mean field* assumption, i.e., by assuming that the weights are independent such that $q(\mathbf{W}|\nu)$ factorizes into a product of factors $q(w|\nu_w)$ for each weight $w \in \mathbf{W}$. The most prominent approach to obtain the variational distribution $q(\mathbf{W}|\nu)$ is by minimizing the KL-divergence $\mathrm{KL}(q(\mathbf{W}|\nu)||p(\mathbf{W}|\mathcal{D}))$ using gradient-based optimization (Ranganath et al., 2014; Blundell et al., 2015).

The Bayesian approach is appealing as distributions over the parameters directly translate into predictive distributions. In contrast to ordinary DNNs that only provide a point estimate prediction, Bayesian neural networks offer predictive uncertainties which are useful to determine how certain the DNN is about its own prediction. Additionally, the Bayesian framework has got several other useful properties that can be exploited to obtain resource-efficient DNNs. For instance, the prior $p(\mathbf{W})$ allows us to incorporate information about properties, such as sparsity, that we expect to be present in the DNN. In Section 3.1.3, we review weight quantization approaches based on the Bayesian paradigm, and in Section 3.2.3, we review pruning approaches based on the Bayesian paradigm.

3. Resource Efficiency in Deep Neural Networks

In this section, we provide a comprehensive overview of methods that enhance the efficiency of DNNs regarding memory footprint, computation time, and energy requirements. We have

identified three different major approaches that aim to reduce the computational complexity of DNNs, i.e., (i) weight and activation quantization, (ii) network pruning, and (iii) structural efficiency. These categories are not mutually exclusive, and we present individual methods in the category where their contribution is most significant.

3.1 Quantized Neural Networks

Quantization in DNNs is concerned with reducing the number of bits used for the representation of the weights and the activations. The reduction in memory requirements are obvious: Using fewer bits for the weights results in a lower memory overhead for storing the corresponding model, and using fewer bits for the activations results in a lower memory overhead for computing predictions. Furthermore, representations using fewer bits often facilitate faster computation. For instance, when quantization is driven to the extreme with binary weights $w \in \{-1, 1\}$ *and* binary activations $x \in \{-1, 1\}$, floating-point or fixed-point dot products are replaced by hardware-friendly logical XNOR and bitcount operations. In this way, a sophisticated DNN is essentially reduced to a logical circuit.

However, training such discrete-valued DNNs[5] is difficult as they cannot be directly optimized using gradient-based methods. The challenge is to reduce the number of bits as much as possible while at the same time keeping the prediction accuracy close to that of a well-tuned full-precision DNN. Subsequently, we provide a literature overview of approaches that train reduced-precision DNNs, and, in a broader view, we also consider methods that use reduced-precision computations during backpropagation to facilitate low-resource training.

3.1.1 EARLY QUANTIZATION APPROACHES

Approaches for reduced-precision computations date back at least to the early 1990s. The two works of Höhfeld and Fahlman (Höhfeld and Fahlman, 1992a; Höhfeld and Fahlman, 1992b) rounded the weights during training to fixed-point format with different numbers of bits. They observed that training eventually stalls as small gradient updates are always rounded to zero. As a remedy, they proposed stochastic rounding, i.e., rounding values to the nearest value with a probability proportional to the distance to the nearest value. These quantized gradient updates are correct in expectation, do not cause training to stall, and yield good performance with substantially fewer bits than deterministic rounding. More recently, Gupta et al. (2015) have shown that stochastic rounding can also be applied to modern deep architectures, as demonstrated on a hardware prototype.

Lin et al. (2015) propose a method to reduce the number of multiplications required during training. At forward propagation, the weights are stochastically quantized to either binary weights $w \in \{-1, 1\}$ or ternary weights $w \in \{-1, 0, 1\}$ to remove the need for multiplications at all. During backpropagation, inputs and hidden neurons are quantized to powers of two, reducing multiplications to cheaper bit-shift operations, and leaving only a negligible number of floating-point multiplications to be computed. However, the speed-up is limited to training since for inference the full-precision weights are required.

Courbariaux et al. (2015a) empirically studied the effect of different numeric formats (i.e., floating-point, fixed-point, and dynamic fixed-point) with varying bit widths on the

---

5. Due to finite precision of computer arithmetic, in fact any DNN is discrete-valued. However, we use this term here to emphasize the extremely small number of values.

performance of DNNs. Lin et al. (2016) consider fixed-point quantization of pre-trained full-precision DNNs. They formulate a convex optimization problem to minimize the total number of bits required to store the weights and the activations under the constraint that the total output signal-to-quantization noise ratio is larger than a certain prespecified value. A closed-form solution of the convex objective yields layer-specific bit widths.

3.1.2 Quantization-aware Training

Quantization operations, being piecewise constant functions with either undefined or zero derivatives, are not applicable to gradient-based learning using backpropagation. In recent years, the STE (Bengio et al., 2013) (see Section 2.6) became the method of choice to compute an approximate gradient for training DNNs with weights that are represented using a very small number of bits. Such methods typically maintain a set of full-precision weights that are quantized during forward propagation. During backpropagation, the gradients are propagated through the quantization functions by assuming that their gradient equals one. In this way, the full-precision weights are updated using gradients computed at the quantized weights. At test-time, the full-precision weights are abandoned and only the quantized reduced-precision weights are kept. We term this scheme *quantization-aware training* since quantization is an essential part during forward-propagation and it is intuitive to think of the real-valued weights as becoming robust to quantization. In a similar manner, many methods employ the STE to approximate the gradient for the quantization of activations.

In Courbariaux et al. (2015b), binary weight DNNs are trained using the STE to get rid of expensive floating-point multiplications. They consider deterministic rounding using the sign function and stochastic rounding using probabilities determined by the hard sigmoid function $\max(0, \min(1, (w+1)/2))$. During backpropagation, a set of auxiliary full-precision weights is updated based on the gradients of the quantized weights. Hubara et al. (2016) extended this work by also quantizing the activations to a single bit using the sign activation function. This reduces the computational burden dramatically as floating-point multiplications and additions are reduced to hardware-friendly logical XNOR and bitcount operations, respectively.

Li et al. (2016) trained ternary weights $w \in \{-a, 0, a\}$. Their quantizer sets weights whose magnitude is lower than a certain threshold Δ to zero, while the remaining weights are set to $-a$ or a according to their sign. Their approach determines $a > 0$ and Δ *during forward propagation* by approximately minimizing the squared quantization error of the real-valued weights. Zhu et al. (2017) extended this work to ternary weights $w \in \{-a, 0, b\}$ where $a > 0$ and $b > 0$ are trainable parameters subject to gradient updates. They propose to select Δ^l based on the maximum full-precision weight magnitude in each layer, i.e., $\Delta^l = t \cdot \max\{|w| : w \in \mathbf{W}^l\}$ with t being a hyperparameter. These asymmetric weights considerably outperform symmetric weights as used by Li et al. (2016).

Rastegari et al. (2016) approximate full-precision weight filters in CNNs by $\mathbf{W} = \alpha \mathbf{B}$ where α is a scalar and $\mathbf{B}$ is a binary weight matrix. This reduces the bulk of floating-point multiplications inside the convolutions to either additions or subtractions and only requires a single multiplication per output neuron with the scalar α. In a further step, the layer inputs $\mathbf{x}^l$ are quantized in a similar way to perform the convolution using only efficient XNOR and bitcount operations, followed by two floating-point multiplications per output

neuron. Again, the STE is used during backpropagation. Lin et al. (2017b) generalized the ideas of Rastegari et al. (2016) by approximating the full-precision weights using linear combinations of *multiple* binary weight filters for improved classification accuracy.

While most activation binarization methods use the sign function which can be seen as an approximation of the tanh function, Cai et al. (2017) proposed a half-wave Gaussian quantization that more closely resembles the predominant ReLU activation function.

Motivated by the fact that weights and activations typically exhibit a non-uniform distribution, Miyashita et al. (2016) proposed to quantize values to powers of two. Their representation allows getting rid of expensive multiplications, and they report higher robustness to quantization than linear rounding schemes using the same number of bits. Zhou et al. (2017) proposed incremental network quantization where the weights of a pre-trained DNN are first partitioned into two sets. The weights in the first set are quantized to either zero or powers of two. The weights in the second set are kept at full precision and retrained to recover from the potential accuracy degradation due to quantization. They iterate partitioning, quantization, and retraining until all weights are quantized.

Jacob et al. (2018) proposed a quantization scheme that accurately approximates floating-point operations using only integer arithmetic to speed up computation. During training, their forward pass simulates the quantization step to keep the performance of the quantized DNN close to the performance of using single-precision. At test-time, weights are represented as 8-bit integer values, reducing the memory footprint by a factor of four.

Liu et al. (2018) introduced Bi-Real net, a ResNet-inspired architecture where the residual path is implemented with efficient binary convolutions while the shortcut path is kept real-valued to preserve the expressiveness of the DNN. The residual in each layer is computed by first transforming the input with the sign activation, followed by a binary convolution, and a final batch normalization step.

Instead of using a fixed quantizer, in *LQ-Net* (Zhang et al., 2018a) the quantizer is adapted during training. The proposed quantizer is inspired by the representation of integers as linear combinations $\mathbf{v}^\top \mathbf{b}$ with $\mathbf{v} = (2^0, \dots, 2^{K-1})$ and $\mathbf{b} \in \{0, 1\}^K$. The key idea is to consider a quantizer that assigns values to the nearest value representable as such a linear combination $\mathbf{v}^\top \mathbf{b}$ and to treat $\mathbf{v} \in \mathbb{R}^K$ as trainable parameters. It is shown that such a quantizer is compatible with efficient bit operations. The quantizer is optimized *during forward propagation* by minimizing the quantization error objective $\|\mathbf{B}\mathbf{v} - \mathbf{x}\|^2$ for $\mathbf{B} \in \{-1, 1\}^{N \times K}$ and $\mathbf{v}$ by alternately fixing $\mathbf{B}$ and minimizing $\mathbf{v}$ and vice versa. It is proposed to use layer-wise quantizers for the activations and channel-wise quantizers for the weights, i.e., an individual quantizer for each layer and channel, respectively.

Relaxed Quantization (Louizos et al., 2019) introduces a stochastic differentiable soft rounding scheme. By injecting additive noise to the deterministic weights before rounding, one can compute probabilities of the weights being rounded to specific values in a predefined discrete set. Subsequently, these probabilities are used to differentiably round the weights using the Gumbel-softmax approximation (Jang et al., 2017). Since this soft rounding scheme produces only values that are close to values from the discrete set but which are not exactly from this set, the authors also propose a hard variant using the STE.

Dong et al. (2019) introduced Hessian-aware mixed-precision quantization for DNNs. Their method quantifies the sensitivity of individual DNN blocks to weight quantization using the largest eigenvalue of the block-wise Hessian matrices which can be computed using

the power iteration method. They compute two different orderings of the individual DNN blocks, both of which are based on these eigenvalues. The first ordering determines a relative ordering of the bit widths of individual blocks. This substantially reduces the exponential search space of layer-specific weights and allows them to manually set appropriate bit widths. The second ordering takes these bit widths into account and determines the sequence in which blocks are quantized and fine-tuned using quantization-aware training.

In general, a linear quantizer has three characteristic properties, (i) a step size Q_d, (ii) a dynamic range $Q_{\max}$, and (iii) the number of bits Q_b. Since these quantities are interrelated according to

$$Q_{\max} = (2^{Q_b-1} - 1)Q_d, \tag{12}$$

it is specified by knowing any two of them (Uhlich et al., 2020). Given fixed layer-wise bit widths Q_b^l, Esser et al. (2020) incorporated layerwise step sizes Q_d^l as trainable parameters in the computation graph. By training the step sizes Q_d^l using the STE, they are adapted to the given objective. This is in contrast to previous work, such as XNOR-Net (Rastegari et al., 2016), that determine the step size Q_d^l using certain statistics obtained from the values to be quantized. Uhlich et al. (2020) extended this idea to mixed-precision quantization. They investigated the three different possibilities to specify a linear quantizer (12) by only two of its characteristic properties and discovered substantial differences in the training behavior. They propose to parameterize the quantizers using the step size Q_d^l and the dynamic range $Q_{\max}^l$ and to train these values using backpropagation and the STE to obtain layerwise bit widths Q_b^l.

There also exist works that perform quantization *during* backpropagation to facilitate resource-efficient training. Zhou et al. (2016) presented several quantization schemes for the weights and the activations that allow for flexible bit widths. Furthermore, they also propose a quantization scheme for backpropagation to facilitate low-resource training. In accordance with earlier work mentioned above, they note that stochastic quantization is essential for their approach. In Wu et al. (2018b), weights, activations, weight gradients, and activation gradients are subject to customized quantization schemes that allow for variable bit widths and facilitate integer arithmetic during training and testing. In contrast to Zhou et al. (2016), the work of Wu et al. (2018b) accumulates weight changes to low-precision weights instead of full-precision weights.

While most work on quantization based approaches is empirical, some works gained more theoretical insights (Li et al., 2017; Anderson and Berg, 2018). The recent work of Shekhovtsov et al. (2020) has shown that for stochastic binary networks the STE arises from particular linearization approximations.

Even for large language models (Touvron et al., 2023; Brown et al., 2020; Shoeybi et al., 2019) quantization can offer benefits. However, for such large models quantization-aware retraining is impractical. Frantar et al. (2023) proposed a one-shot post-training quantization scheme for transformers which supports low bit-width quantization and can compress transformer-based language models within few GPU hours. Targeting the same problem, Lin et al. (2023) introduced activation-aware weight quantization, which exploits the fact that the weights of large language models are not equally important. They propose to guide the selection of important weights by activation magnitudes (rather than weight magnitudes) and protecting salient weights by per-channel scaling.

3.1.3 BAYESIAN APPROACHES FOR QUANTIZATION

In this section, we review some quantization approaches, most of which are closely related to the Bayesian variational inference framework (see Section 2.7).

The work of Achterhold et al. (2018) builds on the variational dropout based pruning approach of Louizos et al. (2017) (see Section 3.2.3). They introduce a mixture of log-uniform priors whose mixtures are centered at predefined quantization values. Consequently, the approximate posterior also concentrates at these values such that weights can be safely quantized without requiring a fine-tuning procedure.

The following works in this section directly operate on *discrete* weight distributions and, consequently, do not require a rounding procedure. Soudry et al. (2014) approximate the true posterior $p(\mathbf{W}|\mathcal{D})$ over discrete weights using expectation propagation (Minka, 2001) with closed-form online updates. Starting with an uninformative approximation $q(\mathbf{W}|\nu)$, their approach combines the current approximation $q(\mathbf{W}|\nu)$ (serving as the prior in Bayes' rule (10)) with the likelihood for a single-sample data set $\mathcal{D}_n = \{(\mathbf{x}_n^0, t_n)\}$ to obtain a refined posterior. By proposing several approximations, they obtain a closed-form refinement step.

Although deviating from the Bayesian variational inference framework as no similarity measure to the true posterior is optimized, the approach of Shayer et al. (2018) trains a distribution $q(\mathbf{W}|\nu)$ over either binary weights $w \in \{-1,1\}$ or ternary weights $w \in \{-1,0,1\}$. They propose to minimize an expected loss $\mathbb{E}_{q(\mathbf{W}|\nu)}[\mathcal{L}(\mathbf{W};\mathcal{D})]$ for the variational parameters ν with gradient-based optimization using the local reparameterization trick (Kingma et al., 2015). After training has finished, the discrete weights are obtained by either sampling or taking a mode from $q(\mathbf{W}|\nu)$. Since their approach is limited to the ReLU activation function, Peters and Welling (2018) extended their work to the sign activation function. This involves several non-trivial changes since the sign activation, due to its zero derivative, requires that the local reparameterization trick must be performed *after* the sign function. Consequently, *distributions* need to be propagated through commonly used building blocks such as batch normalization and pooling operations. Roth et al. (2019) further extended these works to beyond three distinct discrete weights, and they introduced some technical improvements.

Van Baalen et al. (2020) propose a Bayesian mixed-precision quantization method for power-of-two bit widths. Their method is based on a recursive view of quantization where residual quantization errors are repeatedly quantized. They introduce gates that determine how many recursive quantization steps should be performed which in turn determines the number of used bits. While the quantization itself is subject to the STE, they propose to train gate probabilities using the Bayesian variational inference framework. The use of fewer bits for quantization is encouraged using a specific prior and, through an additional zero-bit gate, their framework simultaneously allows for weight pruning.

Havasi et al. (2019) introduced a novel Bayesian compression technique that we present here in this section although it is rather a coding technique than a quantization technique. In a nutshell, their approach first computes a variational distribution $q(\mathbf{W}|\nu)$ over real-valued weights using mean field variational inference and then it encodes a sample from $q(\mathbf{W}|\nu)$ in a smart way. They construct an approximation $\tilde{q}(\mathbf{W})$ to $q(\mathbf{W}|\nu)$ by importance

sampling using the prior $p(\mathbf{W})$ as

$$q(\mathbf{W}\,|\,\nu) \approx \tilde{q}(\mathbf{W}) = \sum_{i=1}^{2^K} \frac{q(\mathbf{W}_i\,|\,\nu)}{p(\mathbf{W}_i)} \delta_{\mathbf{W}_i}(\mathbf{W}) \qquad \text{with} \qquad \mathbf{W}_i \sim p(\mathbf{W}), \qquad (13)$$

where $\delta_{\mathbf{W}_i}$ denotes a point mass located at $\mathbf{W}_i$. In the next step, a sample $\mathbf{W}$ from $\tilde{q}(\mathbf{W})$ (or, equivalently, an approximate sample from $q(\mathbf{W}\,|\,\nu)$) is drawn which can be encoded by the corresponding number $k \in \{1, \ldots, 2^K\}$ using K bits. Using the same random number generator initialized with the same seed as in (13), the weights $\mathbf{W}$ can be recovered by sampling 2^K weights $\mathbf{W}_i$ from the prior $p(\mathbf{W})$ and selecting $\mathbf{W}_k$. Since the number of samples 2^K required to obtain a reasonable approximation to $q(\mathbf{W}\,|\,\nu)$ in (13) grows exponentially with the number of weights, this sampling based compression scheme is performed for smaller weight blocks such that each weight block can be encoded with K bits.

3.2 Network Pruning

Network pruning methods aim to achieve parameter sparsity by setting a substantial number of DNN weights to zero. Subsequently, the sparsity is exploited to improve resource efficiency of the DNN. On the one hand, there exist *unstructured* pruning approaches that set individual weights, regardless of their location in a weight tensor, to zero. Unstructured pruning approaches are typically less sensitive to accuracy degradation, but they require special sparse tensor data structures that in turn yield practical efficiency improvements only for very high sparsity. On the other hand, *structured* pruning methods aim to set whole weight structures to zero, e.g., by setting *all* weights of a matrix column to zero we would effectively prune an entire neuron. Conceptually, structured pruning is equivalent to removing tensor dimensions such that the reduced tensor remains compatible with highly optimized dense tensor operations.

In this section, we start with the unstructured case which includes many of the earlier approaches and continue with structured pruning that has been the focus of more recent works. Then we review approaches that relate to Bayesian principles before we discuss approaches that prune structures dynamically during forward propagation.

3.2.1 Unstructured Pruning

One of the earliest approaches to reduce the network size is the *optimal brain damage* algorithm of LeCun et al. (1989). Their main finding is that pruning based on weight magnitude is suboptimal, and they propose a pruning scheme based on the increase in loss function. Assuming a pre-trained network, a local second-order Taylor expansion with a diagonal Hessian approximation is employed that allows us to estimate the change in loss function caused by weight pruning without re-evaluating the costly network function. Removing parameters is alternated with retraining the pruned network. In this way, the model size can be reduced substantially without deteriorating its performance. Hassibi and Stork (1992) found the diagonal Hessian approximation to be too restrictive, and their *optimal brain surgeon* algorithm uses an approximated full covariance matrix instead. While their method, similar as in LeCun et al. (1989), prunes weights that cause the least increase in loss function, the remaining weights are simultaneously adapted to compensate for the

negative effect of weight pruning. This bypasses the need to alternate several times between pruning and retraining the pruned network.

However, it is not clear whether these approaches scale up to modern DNN architectures since computing the required (diagonal) Hessians is substantially more demanding (if not intractable) for millions of weights. Therefore, many of the more recently proposed techniques still resort to magnitude-based pruning. Han et al. (2015) alternate between pruning connections below a certain magnitude threshold and re-training the pruned DNN. The results of this simple strategy are impressive, as the number of parameters in pruned DNNs is an order of magnitude smaller ($9\times$ for AlexNet and $13\times$ for VGG-16) than in the original networks. Hence, this work shows that DNNs are often heavily over-parameterized. In a follow-up paper, Han et al. (2016) proposed *deep compression*, which extends the work Han et al. (2015) by a parameter quantization and parameter sharing step, followed by Huffman coding to exploit the non-uniform weight distribution. This approach yields a reduction in memory footprint by a factor of 35–49 and, consequently, a reduction in energy consumption by a factor of 3–5.

Guo et al. (2016) discovered that irreversible pruning decisions limit the achievable sparsity and that it is useful to reincorporate weights pruned in an earlier stage. In addition to each dense weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, they maintain a corresponding binary mask matrix $\mathbf{T} \in \{0,1\}^{m \times n}$ that determines whether a weight is currently pruned or not. In particular, the actual weights used during forward propagation are obtained as $\mathbf{W} \odot \mathbf{T}$ where $\odot$ denotes element-wise multiplication. Their method alternates between updating the weights $\mathbf{W}$ based on gradient descent, and updating the weight masks $\mathbf{T}$ by thresholding the real-valued weights according to

$$
T_{i,j}^{t+1} = \begin{cases} 0 & \text{if } |W_{i,j}^t| \in [0,a) \\ T_{i,j}^t & \text{if } |W_{i,j}^t| \in [a,b) \\ 1 & \text{if } |W_{i,j}^t| \in [b,\infty) \end{cases} , \tag{14}
$$

where a and b are thresholds and t refers to the iteration number. Most importantly, weight updates are also applied to the currently pruned weights according to $\mathbf{T}$ using the STE, such that pruned weights can reappear in (14). This reduces the number of parameters of AlexNet by a factor of 17.7 without deteriorating performance.

Gadhikar et al. (2023) provide theoretical insights and have demonstrated empirically that sparsity can also be introduced by using a fixed but randomly selected sparsity pattern. While random pruning is computationally inexpensive, it may not achieve optimal sparsity. However, it still holds potential for further refinement, making it valuable as initial sparse random masks can bypass the computationally expensive process of pruning a dense network from scratch.

3.2.2 STRUCTURED PRUNING

In Mariet and Sra (2016), a determinantal point process (DPP) is used to find a group of neurons that are diverse and exhibit little redundancy. Conceptually, a DPP for a given ground set $\mathcal{S}$ defines a distribution over subsets $S \subseteq \mathcal{S}$ where subsets containing diverse elements have high probability. They consider $\mathcal{S}$ to be the set of N-dimensional vectors that individual neurons compute over the whole data set. Their approach samples a diverse

set of neurons $S \subseteq \mathcal{S}$ according to the DPP and then prunes the other neurons $\mathcal{S} \setminus S$. To compensate for the negative effect of pruning, the outgoing weights of the remaining neurons after pruning are adapted so as to minimize the activation change of the next layer.

Wen et al. (2016) incorporated group lasso regularizers in the objective to obtain different kinds of sparsity in the course of training. They were able to remove filters, channels, and even entire layers in architectures containing skip connections. Liu et al. (2017) proposed to introduce an ℓ^1-norm regularizer on the scale parameters γ of batch normalization and to set $\gamma = 0$ by thresholding. Since each batch normalization parameter γ corresponds to a particular channel in the network, this results in channel pruning with minimal changes to existing training pipelines. In Huang and Wang (2018), the outputs of different structures are scaled with individual trainable scaling factors. By using a sparsity enforcing ℓ^1-norm regularizer on these scaling factors, the outputs of the corresponding structures are driven to zero and can be pruned.

Rather than pruning based on small parameter values, ThiNet (Luo et al., 2017) is a data-driven approach that prunes channels having the least impact on the subsequent layer. To prune channels in layer l, they propose to sample several activations $x_{d,w,h}^{l+1}$ at randomly selected spatial locations (w, h) and channels d of the following layer, and to greedily prune channels whose removal results in the least increase of squared error over these randomly selected activations. After pruning, they adapt the remaining filters to minimize the squared reconstruction error by minimizing a least squares problem.

Louizos et al. (2018) propose to multiply weights with stochastic binary 0-1 gates associated with trainable probability parameters that effectively determine whether a weight should be pruned or not. They formulate an expected loss with respect to the distribution over the stochastic binary gates. By incorporating an expected ℓ^0-norm regularizer over the weights, the probability parameters associated with these gates are encouraged to be close to zero. To enable the use of the reparameterization trick, a continuous relaxation of the binary gates using a modified binary Gumbel-softmax distribution is used (Jang et al., 2017). They show that their approach can be used for structured sparsity by associating the stochastic gates to entire structures such as channels. Li and Ji (2019) extended this work by using the recently proposed unbiased ARM gradient estimator (Yin and Zhou, 2019) instead of using the biased Gumbel-softmax approximation.

While typically a large, over-parameterized network is subject to one or multiple iterations of training, pruning, and re-training, Liu et al. (2019b) argue that for many neural architectures, training on the fixed sparse architecture from scratch yields good results. Their experiments imply that often the sparse structure itself is important, rather than the specific weights that remain after pruning. Furthermore, this highlights a connection between pruning and neural architecture search.

3.2.3 Bayesian Pruning

In Graves (2011) and Blundell et al. (2015), mean field variational inference is employed to obtain a factorized Gaussian approximation $q(\mathbf{W} | \nu)$, i.e., instead of learning a deterministic weight w per connection, they train for each connection a weight mean w_μ and a weight variance w_{σ^2}. After training, weights are pruned by thresholding the "signal-to-noise ratio" $|w_\mu / w_\sigma|$.

Some pruning approaches are based on variational dropout (Kingma et al., 2015) which interprets dropout as performing variational inference with specific prior and approximate posterior distributions. Within this framework, the otherwise fixed dropout rates α of Gaussian dropout appear as free parameters that can be optimized to improve a variational lower bound. Molchanov et al. (2017) exploited this freedom to optimize individual weight dropout rates w_α such that weights w can be safely pruned if their dropout rate w_α is large. This idea has been extended by Louizos et al. (2017) by using sparsity enforcing priors and assigning dropout rates to groups of weights that are all connected to the same structure, which in turn allows for structured pruning. Furthermore, they show how their approach can be used to determine an appropriate bit width for each weight by exploiting the well-known connection between Bayesian inference and the minimum description length (MDL) principle (Grünwald, 2007).

3.2.4 Dynamic Network Pruning

So far, we have presented methods that result in a fixed reduced architecture. In the following, we present methods that determine dynamically in the course of forward propagation which structures should be computed or, equivalently, which structures should be pruned. The intuition behind this idea is to vary the time spent for computing predictions based on the difficulty of the given input samples $\mathbf{x}^0$.

Lin et al. (2017a) proposed to train, in addition to the DNN, a recurrent neural network (RNN) as decision network which determines the channels to be computed using reinforcement learning. To keep the decision network lightweight and fast, it is only fed with compressed feature maps using global pooling. Thus, the RNN can aggregate state information over the layers to compute its pruning decisions, saving enough computations to compensate for the overhead and thus being more efficient than static alternatives.

In Dong et al. (2017), convolutional layers of a DNN are extended by a parallel low-cost convolution whose output after the ReLU function is used to scale the outputs of the potentially high-cost convolution. Due to the ReLU function, several outputs of the low-cost convolution will be exactly zero such that the computation of the corresponding output of the high-cost convolution can be omitted. For the low-cost convolution, they propose to use weight tensors $\mathbf{W} \in \mathbb{R}^{1 \times 1 \times C \times D}$ and $\mathbf{W} \in \mathbb{R}^{K \times K \times C \times 1}$. However, practical speed-ups are only reported for the $K \times K$ convolution where all channels at a given spatial location might get set to zero.

In a similar approach proposed by Gao et al. (2019), the spatial dimensions of a feature map are reduced by global average pooling to a vector $\mathbf{a} \in \mathbb{R}^C$ which is linearly transformed to $\mathbf{b} \in \mathbb{R}^D$ using a single low-cost fully connected layer. To obtain a sparse vector $\mathbf{c} \in \mathbb{R}^D$, $\mathbf{b}$ is fed into the ReLU function, followed by a k-winner-takes-all function that sets all entries of a vector to zero that are not among the k largest entries in absolute value. By multiplying $\mathbf{c}$ in a channel-wise manner to the output of a high-cost convolution, at least $D - k$ channels will be zero and need not be computed. The number of channels k is derived from a predefined minimal pruning ratio hyperparameter.

3.3 Structural Efficiency in DNNs

In this section, we review strategies that establish certain structural properties in DNNs to improve computational efficiency. Each of the proposed subcategories in this section follows rather different principles and the individual techniques might not be mutually exclusive.

3.3.1 Weight Sharing

Another technique to reduce the model size is weight sharing. We note that weight sharing and quantization methods (see Section 3.1) are closely related: Quantization methods often have an inherent weight sharing property since the number of possible quantization values is often much smaller than the number of weights. However, the purpose of a method is typically different depending on which category it belongs to. On the one hand, the focus of weight quantization methods typically lies on the employed numerical formats. The purpose of these formats is to reduce the storage per weight and to facilitate more efficient computations. Furthermore, the number of distinct weight values is typically rather small and fixed, and the particular weight values are often constrained or even fixed in advance. On the other hand, the purpose of weight sharing is to reduce the memory by reducing the overall number of distinct weight values. For these methods, the particular weight values typically remain unconstrained. Note that some methods cannot be clearly attributed to either category, e.g., in deep compression (Han et al., 2016) weight sharing and quantization are in part used synonymously.

Chen et al. (2015) used a hashing function to randomly group network connections into "buckets", where the connections in each bucket share the same weight value. The advantage of their approach is that weight assignments need not be stored explicitly since they are given implicitly by the hashing function. The authors show a memory footprint reduction by a factor of 10 while keeping the prediction quality essentially unaffected.

Ullrich et al. (2017) extended the soft weight sharing approach proposed by Nowlan and Hinton (1992) to achieve both weight sharing and sparsity. The idea is to select a Gaussian mixture model prior over the weights and to train both the weights as well as the parameters of the mixture components. During training, the mixture components collapse to point measures and each weight gets attracted by a certain weight component. After training, weight sharing is obtained by assigning each weight to the mean of the component that best explains it, and weight pruning is obtained by assigning a relatively high mixture mass to a component with a fixed mean at zero.

Roth and Pernkopf (2018) utilized weight sharing to reduce the memory footprint of a large Bayesian ensemble of DNNs. The weight sharing is enforced by introducing a Dirichlet process prior over the weight prior distribution. They propose a sampling based inference scheme by alternately sampling weight assignments using Gibbs sampling and sampling weights using Hamiltonian Monte Carlo (Neal, 1992). By using the same weight assignments for multiple weight samples, the memory overhead for the weight assignments becomes negligible and the total memory footprint of an ensemble is reduced.

3.3.2 Knowledge Distillation

Knowledge distillation is a method where the knowledge contained in a large *teacher* model is transferred to a smaller *student* model. In the first step, a large teacher model is obtained

with conventional training methods on the given training data. Subsequently, the smaller student model is trained on data where the ground truth labels have been replaced by the soft labels obtained from the output of the teacher model, e.g., from the softmax output of a DNN. It has been shown that this substantially increases the accuracy of the student model compared to directly training on the given training data.

This general scheme is model agnostic, and early works applied knowledge distillation to compress *ensembles* of shallow neural networks (Zeng and Martinez, 2000) and other types of classifiers (Bucila et al., 2006) into a single neural network. Zeng and Martinez (2000) have shown that training on soft labels obtained from the teacher results in higher accuracy than training on the actual hard predictions. The work of Bucila et al. (2006) emphasizes the ability to train the student on unlabeled data to further reduce the accuracy gap between student and teacher. In addition, they presented a method to generate new synthetic inputs from the given training set, which might be useful if additional unlabeled data is limited or not available. They showed that the accuracy of the student can improve substantially when trained on these synthetically generated inputs.

Ba and Caruana (2014) applied these ideas to investigate the importance of depth in a DNN. They trained shallower (but not necessarily smaller) neural networks by mimicking the output activations $\mathbf{a}^L$ produced by a teacher DNN before applying the softmax function. The resulting shallow models perform similar to their deeper counterparts which was not achievable by training the shallow model on the ground truth targets directly. Therefore, the authors conclude that shallower models are as expressive as deeper models but they are more difficult to train.

The work of Li et al. (2014) and Hinton et al. (2015) applied knowledge distillation with the main focus on reducing model complexity of a large teacher DNN. Hinton et al. (2015) proposed to obtain the soft labels $\hat{\mathbf{y}}$ from the teacher by scaling the output activations with a temperature $\tau > 0$ as

$$\hat{y}_i = \frac{\exp(a_i^L/\tau)}{\sum_j \exp(a_j^L/\tau)}. \tag{15}$$

For $\tau > 1$, the labels tend to become more uniform which has been reported to facilitate training. Furthermore, they propose to utilize the ground truth labels by minimizing a weighted average of the traditional cross-entropy loss based on the ground truth labels t and the knowledge distillation loss based on the soft targets $\hat{\mathbf{y}}$ in (15). Notably, it was the work of Hinton et al. (2015) that coined the term *knowledge distillation*.

FitNets (Romero et al., 2015) extend these ideas by also transferring knowledge from intermediate layers. They select an intermediate layer from the teacher DNN as the *hint layer* which they try to mimic in an intermediate *guide layer* of the student DNN. Since the hint layer and the guide layer are generally of different size, they introduce a regressor that predicts the hint layer from the guide layer. This ensures that the guide layer contains the same information as the hint layer. The proposed procedure operates in two stages. In the first stage, the student is trained up to the guide layer by minimizing the discrepancy between them. In the second stage, the whole student DNN is trained using conventional knowledge distillation as in Hinton et al. (2015).

Kim et al. (2018) argue that matching the raw features of certain intermediate layers as in Romero et al. (2015) is suboptimal since it is difficult to compare individual layers of

different DNNs. Therefore, they propose a method to match more understandable *factors* extracted from the intermediate layers of the student and the teacher DNNs. Starting from a pre-trained teacher DNN, they first train an autoencoder which they call *paraphraser* to extract understandable factors from a selected intermediate layer of the teacher DNN. The student DNN is extended by a regressor which they call *translator* whose purpose is to predict the paraphraser factors from the features of a selected intermediate layer. The student DNN is then trained to simultaneously minimize the cross-entropy loss on the ground truth labels and the difference between paraphraser and translator output. They employ the paraphraser and the translator after the last convolutional layer in their DNNs.

In the context of quantization, knowledge distillation has been used to reduce the accuracy gap between real-valued DNNs and quantized DNNs (Mishra and Marr, 2018; Polino et al., 2018). In particular, a real-valued teacher DNN is used to improve the accuracy of a quantized student DNN. Mishra and Marr (2018) showed improved results using three different modes of knowledge distillation training, including a mode where the student and the teacher are trained simultaneously from scratch.

Phuong and Lampert (2019) transferred knowledge between different parts of the *same* model. They employ multi-exit architectures which provide anytime predictions after certain intermediate layers; therefore, allowing for a trade-off between accuracy and prediction latency at run-time. The knowledge from the (most accurate) final layer is transferred to the earlier exits to improve their accuracy. Furthermore, they show that the earlier layers can be trained with unlabeled data in a semi-supervised setting.

Kaliamoorthi et al. (2021) demonstrated the effectiveness of knowledge distillation for large language models. They proposed a new efficient student architecture consisting of projection, bottleneck, convolution and pooling layers. This projection-based embedding-free architecture is lightweight in terms of latency and parameter footprint.

In a Bayesian context, Korattikara et al. (2015) applied knowledge distillation to condense a large ensemble of DNNs, for instance, obtained by sampling from the posterior distribution $p(\mathbf{W}|\mathcal{D})$. In this way, expected predictions (11) obtained by averaging the outputs of the individual models can be transferred to a single DNN. Their method trains a single student DNN using the outputs of teacher DNNs that are generated on the fly using SGLD (Welling and Teh, 2011).

3.3.3 SPECIAL MATRIX STRUCTURES

In this section, we review approaches that aim to reduce the model size by employing efficient matrix representations. There exist several methods using low-rank decompositions which represent a large matrix (or a large tensor) using only a fraction of the parameters. In most cases, the implicitly represented matrix is never computed explicitly such that also a computational speed-up is achieved. Furthermore, there exist approaches using special matrices that are specified by only few parameters and whose structure allows for extremely efficient matrix multiplications.

Denil et al. (2013) proposed a method that is motivated by training only a subset of the weights and predicting the values of the other weights from this subset. In particular, they represent weight matrices $\mathbf{W} \in \mathbb{R}^{m \times n}$ using a low-rank approximation $\mathbf{UV}$ with $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{V} \in \mathbb{R}^{k \times n}$, and $k < \min\{m, n\}$ to reduce the number of parameters. Instead of learning

both factors $\mathbf{U}$ and $\mathbf{V}$, prior knowledge, such as smoothness of pixel intensities in an image, is incorporated to compute a fixed $\mathbf{V}$ using kernel-techniques or auto-encoders, and only the factor $\mathbf{U}$ is learned.

In Novikov et al. (2015), the tensor train matrix format is employed to substantially reduce the number of parameters required to represent large weight matrices of fully connected layers. Their approach enables the training of very large fully connected layers with relatively few parameters, and they achieve improved performance compared to simple low-rank approximations.

Denton et al. (2014) propose specific low-rank approximations and clustering techniques for individual layers of pre-trained CNNs to reduce both memory footprint and computational overhead. Their approach yields substantial improvements for both the computational bottleneck in the convolutional layers and the memory bottleneck in the fully connected layers. By fine-tuning after applying their approximations, the performance degradation is kept at a decent level. Jaderberg et al. (2014) propose two different methods to approximate pre-trained CNN filters as combinations of rank-1 basis filters to speed up computation. The rank-1 basis filters are obtained either by minimizing a reconstruction error of the original filters or by minimizing a reconstruction error of the outputs of the convolutional layers. Lebedev et al. (2015) approximate the convolution tensor using the canonical polyadic (CP) decomposition—a generalization of low-rank matrix decompositions to tensors—using non-linear least squares. Subsequently, the convolution using this low-rank approximation is performed by four consecutive convolutions, each with a smaller filter, to reduce the computation time substantially.

In Cheng et al. (2015), the weight matrices of fully connected layers are restricted to circulant matrices $\mathbf{W} \in \mathbb{R}^{n \times n}$, which are fully specified by only n parameters. While this dramatically reduces the memory footprint of fully connected layers, circulant matrices also facilitate faster computation as matrix-vector multiplication can be efficiently computed using the fast Fourier transform. In a similar vein, Yang et al. (2015) reparameterize matrices $\mathbf{W} \in \mathbb{R}^{n \times n}$ of fully connected layers using the Fastfood transform as $\mathbf{W}=\mathbf{SHG\Pi HB}$, where $\mathbf{S}$, $\mathbf{G}$, and $\mathbf{B}$ are diagonal matrices, $\mathbf{\Pi}$ is a random permutation matrix, and $\mathbf{H}$ is the Walsh-Hadamard matrix. This reparameterization requires only a total of $4n$ parameters, and similar as in Cheng et al. (2015), the fast Hadamard transform enables an efficient computation of matrix-vector products.

3.3.4 MANUAL ARCHITECTURE DESIGN

Instead of modifying existing architectures to make them more efficient, manual architecture design is concerned with the development of new architectures that are inherently resource-efficient. Over the past years, several design principles and building blocks for DNN architectures have emerged that exhibit favorable computational properties and sometimes also improve performance.

CNN architectures are typically designed to have a transition from convolutional layers to fully connected layers. At this transition, activations at all spatial locations of each channel are typically used as individual input features for the following fully connected layer. Since the number of these features is typically large, there is a memory bottleneck for storing the parameters of the weight matrix especially in the first fully connected layer.

Lin et al. (2014a) introduced two concepts that have been widely adopted by subsequent works. The first technique, *global average pooling*, largely solves the above-mentioned memory issue at the transition to fully connected layers. Global average pooling reduces the spatial dimensions of each channel into a single feature by averaging over all values within a channel. This reduces the number of features at the transition drastically, and by having the same number of channels as there are classes, it can also be used to completely remove fully connected layers. Secondly, they used 1×1 convolutions with weight kernels $\mathbf{W} \in \mathbb{R}^{1 \times 1 \times C \times D}$ which can be seen as performing the operation of a fully connected layer over each spatial location across all channels.

These 1×1 convolutions have been adopted by several popular architectures (Szegedy et al., 2015; He et al., 2016; Huang et al., 2017) and, due to their favorable computational properties compared to convolutions that take a spatial neighborhood into account, later have also been exploited to improve computational efficiency. For instance, InceptionNet (Szegedy et al., 2015) proposed to split standard $K \times K$ convolutions into two cheaper convolutions: (i) a 1×1 convolution to reduce the number of channels such that (ii) a subsequent $K \times K$ convolution is performed faster. Similar ideas are used in *SqueezeNet* (Iandola et al., 2016) which employs 1×1 convolutions to reduce the number of input channels of subsequent parallel 1×1 and 3×3 convolutions. In addition, SqueezeNet uses the global average pooling output of per-class channels directly as input to the softmax in order to avoid fully connected layers that typically consume the most memory. On top of that, they also applied deep compression (Han et al., 2016) (see Section 3.2.1) to reduce the memory footprint of their model even further.

Szegedy et al. (2016) extended the InceptionNet architecture by spatially separable convolutions to reduce the computational complexity, i.e., a $K \times K$ convolution is split into a $K \times 1$ convolution followed by a $1 \times K$ convolution. In *MobileNet* (Howard et al., 2017a) depthwise separable convolutions are used to split a standard convolution in another way: (i) a depthwise convolution and (ii) a 1×1 convolution. The depthwise convolution applies a $K \times K$ filter to each channel separately without taking the other channels into account whereas the 1×1 convolution then aggregates information across channels. Although these two cheaper convolutions together are less expressive than a standard convolution, they can be used to trade off a small loss in prediction accuracy with a drastic reduction in computational overhead and memory requirements.

Sandler et al. (2018b) extended these ideas in their *MobileNetV2* to an architecture with residual connections. A typical residual block with bottleneck structure in ResNet (He et al., 2016) contains a 1×1 bottleneck convolution to reduce the number of channels, followed by a 3×3 convolution, followed by another 1×1 convolution to restore the original number of channels again. Contrary to that building block, MobileNetV2 introduces an *inverted* bottleneck structure where the shortcut path contains the bottleneck and the residual path performs computations in a high-dimensional space. In particular, the residual path performs a 1×1 convolution to *increase* the number of channels, followed by a cheap *depthwise* 3×3 convolution, followed by another 1×1 convolution to reduce the number of channels again. They show that their inverted structure is more memory efficient since the shortcut path, which needs to be kept in memory during computation of the residual path, is considerably smaller. Furthermore, they show improved performance compared to the standard bottleneck structure.

While it was more of a technical detail rather than a contribution on its own, AlexNet (Krizhevsky et al., 2012) used *grouped convolutions* with two groups to facilitate model parallelism for training on two GPUs with relatively low memory capacity. Instead of computing a convolution using a weight tensor $\mathbf{W} \in \mathbb{R}^{K \times K \times gC \times gD}$, a grouped convolution splits the input into g groups of C channels that are independently processed using weight tensors $\mathbf{W}_g \in \mathbb{R}^{K \times K \times C \times D}$. The outputs of these g convolutions are then stacked again such that the same number of input and output channels are maintained while considerably reducing the computational overhead and memory footprint.

Although this reduces the expressiveness of the convolutional layer since there is no interaction between the different groups, Xie et al. (2017) used grouped convolutions to enlarge the number of channels of a ResNet model which resulted in accuracy gains while keeping the computational complexity of the original ResNet model approximately the same. Zhang et al. (2018b) introduced a ResNet-inspired architecture called *ShuffleNet* which employs 1×1 *grouped* convolutions since 1×1 convolutions have been identified as computational bottlenecks in previous works, e.g., see Howard et al. (2017a). To combine the computational efficiency of grouped convolutions with the expressiveness of a full convolution, ShuffleNet incorporates *channel shuffle* operations after grouped convolutions to partly recover the interaction between different groups.

Over-parameterization is a typical concern for many vision tasks, and architectural efficiency and compression support progress in learning. This is in contrast to most natural language tasks where the trend is towards ever larger models. For large language models like PaLM (Chowdhery et al., 2023) where 6144 TPUs are used for training, efficient scheduling and communication between hardware resources is essential.

3.3.5 Neural Architecture Search

Neural architecture search (NAS) is a recently emerging field concerned with the automatic discovery of good DNN architectures. This is achieved by designing a discrete space of possible architectures in which we subsequently search for an architecture that optimizes some objective—typically the validation error. By incorporating a measure of resource efficiency into this objective, this technique has recently attracted attention for the automatic discovery of resource-efficient architectures.

The task is very challenging: On the one hand, evaluating the validation error is time-consuming as it requires a full training run and typically only results in a noisy estimate thereof. On the other hand, the space of architectures is typically of exponential size in the number of layers. Hence, the space of architectures needs to be carefully designed in order to facilitate an efficient search within it.

The influential work of Zoph and Le (2017) introduced a scheme to encode DNN architectures of arbitrary depth as sequences of tokens which can be sampled from a controller RNN. This controller RNN is trained with reinforcement learning to generate well performing architectures using the validation error on a held-out validation set as a reward signal. However, the training effort is enormous since more than 10,000 training runs are required to achieve state-of-the-art performance on CIFAR-10. This would be impractical on larger data sets such as ImageNet which was partly solved by subsequent NAS approaches, e.g.,

in Zoph et al. (2018). In this review, we highlight methods that also consider resource efficiency constraints for NAS.

In MnasNet (Tan et al., 2018), a RNN controller is trained by also considering the latency of the sampled DNN architecture measured on a real mobile device. They achieve performance improvements under predefined latency constraints on a specific device. To run MnasNet on the large-scale data sets ImageNet and COCO (Lin et al., 2014b), their algorithm is run on a *proxy task* by only training for five epochs, and only the most promising DNN architectures were trained using more epochs.

Wang et al. (2019) determined the individual bit widths of mixed-precision quantization using a similar reinforcement learning framework. Their controller DNN generates for each layer two bit widths, one for the weights and one for the activations. A pre-trained full-precision DNN is then quantized using these bit widths and fine-tuned for one epoch to obtain a reward signal that is subsequently used to update the controller. Their method incorporates hardware-specific constraints, such as latency and energy consumption, that must be met by the controller.

Instead of generating architectures using a controller, ProxylessNAS (Cai et al., 2019) uses a heavily over-parameterized model where each layer contains several parallel paths, each computing a different architectural block with its individual parameters. For each layer, probability parameters for selecting a particular architectural block are introduced which are trained via backpropagation using the STE. After training, the most probable path determines the selected architecture. To favor resource-efficient architectures, a latency model is build using measurements done on a specific real device whose predicted latencies are used as a differentiable regularizer in the cost function. In their experiments, they show that different target devices prefer individual DNN architectures to obtain a low latency.

Instead of using a different path for different operations in each layer, single-path NAS (Stamoulis et al., 2019) combines all operations in a single *shared weight superblock* such that each operation uses a subset of this superblock. A weight-magnitude-based decision using trainable threshold parameters determines which operation should be performed, allowing for gradient-based training of both the weight parameters and the architecture. Again, the STE is employed to backpropagate through the threshold function.

Wu et al. (2018a) performed mixed-precision quantization using similar NAS concepts to those used by Liu et al. (2019a) and Cai et al. (2019). They introduce gates for every layer that determine the number of bits used for quantization, and they perform continuous stochastic optimization of probability parameters associated with each of these gates.

Liu et al. (2019b) have replicated several experiments of pruning approaches (see Section 3.2) and they observed that the typical workflow of training, pruning, and fine-tuning is often not necessary and only the discovered sparsity structure is important. In particular, they show for several pruning approaches that randomly initializing the weights after pruning and training the pruned structure from scratch results in most cases in a similar performance as performing fine-tuning after pruning. They conclude that network pruning can also be seen as a paradigm for architecture search.

Tan and Le (2019) proposed EfficientNet which employs NAS for finding a resource-efficient architecture as a key component. In the first step, they perform NAS to discover a small resource-efficient model which is much cheaper than searching for a large model directly. In the next step, the discovered model is enlarged by a principled compound scaling

approach which simultaneously increases the number of layers, the number of channels, and the spatial resolution. Although this approach is not targeting resource efficiency on its own, EfficientNet achieves state-of-the-art performance on ImageNet using a relatively small model.

Krieger et al. (2022) proposed Galen, a NAS framework to search effective layer-specific compression parameters for quantization and pruning by considering experimentation on hardware targets. Thereby selected parameters are tested for the task at hand on hardware architectures, and feedback in terms of inference latency is used in the optimization objective. Moreover, a sensitivity analysis is used as proxy to estimate the impact of compression on prediction accuracy. This enables an automated compression of models tailored to given hardware targets balancing multiple compression methods.

Elsken et al. (2019) provides a detailed overview of neural architectures search methods highlighting the key dimensions of search space, search strategy and strategies to estimate performance.

4. Embedded Hardware for Deep Neural Networks

Improvements in hardware for deep learning are a key driver for the recent success stories of AI applications through DNNs. Both training and inference have extremely high demands on their targeted platform and certain hardware requirements can be the deciding factor whether an application can be realized. This section briefly introduces the most important hardware for deep learning and discusses their potentials and limitations. While this discussion is generic and independent from training or inference, it should be noted that all processor concepts are available in different scales, ranging from mobile to server variants.

4.1 CPUs

CPUs were originally designed to optimize single-thread performance in order to execute an individual computation within the shortest possible latency. Unfortunately, single-thread performance is stagnating since the end of Dennard scaling (Dennard et al., 1974), and now performance scaling usually requires parallelization. While multithreading is a rather obvious solution for parallelization that is applicable to many tasks, vectorization is a technique that promises great potential for certain applications. Vectorization applies a single instruction to multiple pre-selected data elements and, thus, avoids costly at-runtime dependency checking while maximizing instruction reuse. CPUs show excellent properties of exploiting sparse DNNs due to their short vector units and the low amount of multithreading together with high frequency. Furthermore, they usually support 8-bit integer formats and feature certain instructions for extremely low representation and, consequently, they are well suited for quantization operations.

4.2 GPUs

GPUs were initially designed to accelerate image and video processing only and are nowadays the most popular general-purpose accelerators for many tasks, such as scientific and AI computations. The architecture consists of many streaming multiprocessors which are highly parallel and each implements many lightweight cores. Thus, GPUs are massively par-

allel processors with large memory that provides extremely high bandwidth and throughput, but significantly lower frequency in comparison to CPUs. The extremely high amount of parallelism and the resulting demand on structured computations, however, virtually prevents the deployment of sparse computations. Modern GPU designs and their respective software stack implement support for reduced-precision computations, such as 8-bit integer and half-precision floating-point formats, which are very well suited for deep learning. More extreme forms of quantization are not yet supported and do not result in more efficient inference or training.

4.3 FPGAs

Field-programmable gate arrays (FPGAs) are a family of processors that implement a large array of configurable logic blocks which can be programmed using hardware description languages (e.g., VHDL, Verilog, HLS). This concept is the main difference to ASICs in terms of technology since the hardware can be designed for specific functional or application requirements. While this reconfigurability enables various opportunities that go beyond the capabilities of CPUs and GPUs, it comes at the cost of much lower frequency and reduced on-chip memory. FPGAs are in principle very well suited for DNNs, since compute units can be specifically tailored to fit the diverse computations while also enabling massive amounts of parallelism. Reconfigurable hardware is especially interesting for compressed DNNs due to their flexibility to implement any data format as well as sparse logic.

4.4 Domain-Specific Accelerators

Recent interest in deep learning has motivated to push advancements in the development of custom accelerators, such as Google's TPUs and Graphcore's IPU. The key feature of the TPU (and most of the other deep learning accelerators) is a 256×256 matrix-multiplication unit that is referred to as *systolic array*. Systolic arrays are a variant of massively parallel processor arrays that are very suitable for regular tasks, such as linear algebra operations, and a promising candidate to address the increasing costs of data movements. The objective of such arrays is to minimize instruction fetch and data access costs by constraining the data flow to matrix and vector operations. However, data movements can only be reduced if locality effects are sufficiently exploited and the data flow constraints of a systolic array may result in poor utilization and latency increase. Such domain-specific accelerators are usually highly constrained when aiming to optimize DNNs through compression. For instance, the TPU supports 8-bit integer and half-precision floating-point formats while other (potentially lower-precision) representations are not efficiently supported by hardware. Furthermore, the dense structure of the systolic arrays demands for similarly dense computations and cannot exploit fine-grained sparsity patterns.

4.5 Loop-Back vs. Data-Flow Architectures

One can roughly categorize hardware platforms for deep learning inference into loop-back and data-flow architectures. Loop-back architectures use a fixed processor and memory system to move data from off-chip memory to the processor and leverage the available compute resources. This is performed for each layer or operation sequentially until inference compu-

tation has finished. The drawback of loop-back architectures is that they potentially require many data movements from and to off-chip memory, which is time and energy consuming. CPUs aim to reduce these memory accesses by featuring large on-chip caches and reuse data as much as possible. Similar are domain-specific accelerators, such as TPUs, which usually feature a large and programmable scratch pad memory on chip. On the contrary, GPUs feature large register files and aim to hide memory latency by leveraging parallel slackness. Another critical aspect of loop-back architectures is low compute utilization, which can potentially occur if certain layer or operation types do not fit the static compute array (i.e., if operation size is too low). The advantage of such a *generic* compute architecture is that they allow arbitrary operations in combination with productive code generation since the hardware does not need to be optimized for a certain task. Continuous improvements in semi-conductor and processor technology are the main improvement factor of such inference engines.

In contrast to this, data-flow architectures use a reconfigurable processor and memory system for computing the inference. Here, each layer or operation within a neural architecture is assigned a dedicated compute engine and its own memory subsystem in order to enable inference in a pipelined fashion. This avoids off-chip accesses for intermediate operations completely by simple forwarding the computed results to the next hardware layer. Furthermore, data-flow architectures achieve excellent utilization of the available hardware logic, since several compute engines can be tailored to the required operation type and latency. One drawback of this data-flow architecture is, however, that it requires long development costs because it does not only require software but also hardware optimizations. In addition, reconfigurable hardware comes at the cost of reduced absolute compute power in comparison to ASIC designs. The main limitation of data-flow architectures is that they demand the entire neural architecture (weights and activations) to be stored on chip, which is highly restrictive for large models.

5. Experimental Results

We provide experimental results for modern DNN architectures trained on well-known benchmark data sets. The focus of our experiments is on quantization (see Section 3.1) and structured pruning approaches (see Section 3.2.2) since they are among the earliest and most efficient approaches to enhance the computational efficiency of DNNs.

We compare several quantization approaches discussed in this paper in terms of prediction quality in Section 5.1.1. Next, we compare different DNN architectures and pruning structures (i.e., the type of structure considered for pruning, such as channels) using model metrics such as number of computations and memory requirements in Section 5.1.2. Furthermore, we benchmark the compressed models on mobile variants of CPU (Section 5.2.1), FPGA (Section 5.2.2), and GPU (Section 5.2.3), and provide an overall comparison in Section 5.2.4.

While domain-specific accelerators, such as Google's TPU, excel in their specific performance, they are usually limited to a set of specific operations and are neither flexible in terms of data types nor sparse calculations. Furthermore, in particular for the TPU, experimentation is often hindered due to limitations in the tool chain which is not flexible

enough to support such optimizations. They are not suited to execute generic compressed models and are therefore not included in the following experiments.

Similarly, there are tools such as Tensorflow Lite (TFLite) which is a library to deploy neural architectures on resource-constrained devices such as mobile or embedded devices. Being a library severely limits the possibility to implement different optimization techniques with regard to resource efficiency. We therefore chose to not cover TFLite in this work. Instead, the interested reader can find an excellent overview of the practical use of TFLite for mobile phones (as well as another review on resource efficiency) in Menghani (2023).

The main focus of this section is to showcase the difficulty of finding good trade-offs between prediction quality and resource efficiency, which is formed by the combination of representational efficiency and computational efficiency. As this paper is mainly dedicated to giving a comprehensive literature overview of the current state of the art, an extensive evaluation of the many presented methods in Section 3 would be infeasible and it is also not within the scope of this paper.

5.1 Prediction Quality of Compressed DNNs

5.1.1 Prediction Quality using Different Quantization Approaches

In the first experiment we compare the performance of several quantization approaches. We use a DenseNet architecture (Huang et al., 2017) consisting of 100 layers with bottleneck and compression layers, i.e., a DenseNet-BC-100. We select the default growth rate of $k = 12$ for the model, i.e., the number of feature maps added per layer. We conduct our experiments on the CIFAR-100 data set where the task is to classify RGB images of size 32×32 pixels to one of 100 object categories. The CIFAR-100 data set is split into 50,000 training images and 10,000 test images. We selected some of the most popular quantization approaches (see Section 3.1) for our comparison: binary weight networks (BWN) (Courbariaux et al., 2015b), binarized neural networks (BNN) (Hubara et al., 2016), DoReFa-Net (Zhou et al., 2016), trained ternary quantization (TTQ) (Zhu et al., 2017), and LQ-Net (Zhang et al., 2018a). For this experiment, we quantize the DNNs in three different modes: (i) weight-only, (ii) activation-only, and (iii) combined weight and activation quantization. However, note that some quantization approaches are designed for a particular mode, e.g., BWN and TTQ only consider weight quantization whereas BNN only considers combined weight and activation quantization.

Figure 3 reports the test errors for different bit widths of the selected quantization approaches. The horizontal red line shows the test error of the real-valued baseline DenseNet-BC-100. For combined weight and activation quantization we use the same bit widths for the weights and the activations.

As expected, the test error decreases gradually with increasing bit widths for all quantization modes and for all quantization approaches. Furthermore, the results indicate that prediction performance is more sensitive to activation quantization than to weight quantization, which is in line with the results reported by many works reviewed in Section 3.1.

The more advanced LQ-Net approach clearly outperforms the rather simple linear quantization of DoReFa-Net and the specialized binary and ternary approaches. However, this performance improvement comes at the cost of longer training times. For instance, the training time per iteration increases—in relation to training without quantization—for DoReFa-
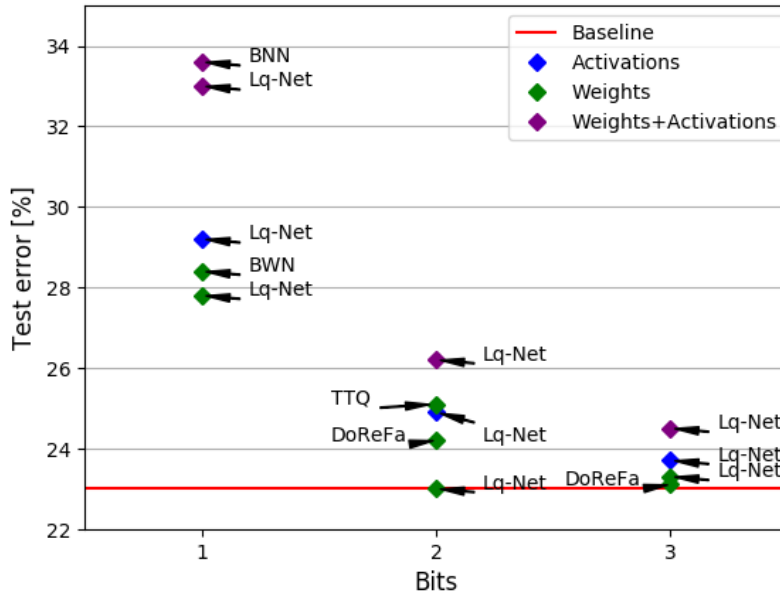
Figure 3: Comparison of several popular quantization approaches (see Section 3.1) using the DenseNet-BC-100 architecture trained on the CIFAR-100 data set. The horizontal red line shows the error of the real-valued baseline. Quantization is performed using different bit widths in three different modes: activation-only (blue), weight-only (green), and combined weight and activation quantization (purple).

Net by a factor of 1.5 compared to a factor of up to 4.6 (depending on the bit width) for LQ-Net.

5.1.2 Prediction Quality using Different Pruning Structures

In the next experiment, we explore the performance metrics of different DNN architectures (ResNet and DenseNet) and pruning structures (such as channels, kernels and groups) on the CIFAR-10 task. CIFAR-10 is similar to CIFAR-100 used in the previous section (i.e., image size and size of training and test sets are equal) except that it contains only ten object classes. We use wide residual networks (WRNs) by Zagoruyko and Komodakis (2016) with a depth of 28 layers, one of the best performing architectures on this task. This architecture is identical to the original ResNet model except that it is scaled in width rather than depth. Additionally, we create a DenseNet variant for this experiment which is scaled in depth to 28 layers and the width is varied until it approximately matches the number of parameters and computations of the WRN model in order to guarantee a fair comparison. We apply parameterized structured pruning (PSP) by Schindler et al. (2020), a method that allows to dynamically learn the shape of DNNs through structured sparsity. PSP parameterizes arbitrary structures in a weight tensor and leverages weight decay to detect unimportant structures that can be pruned. In this experiment, we select pruning structures that are in line with commonly used DNN libraries for convolutions: we use channel pruning to
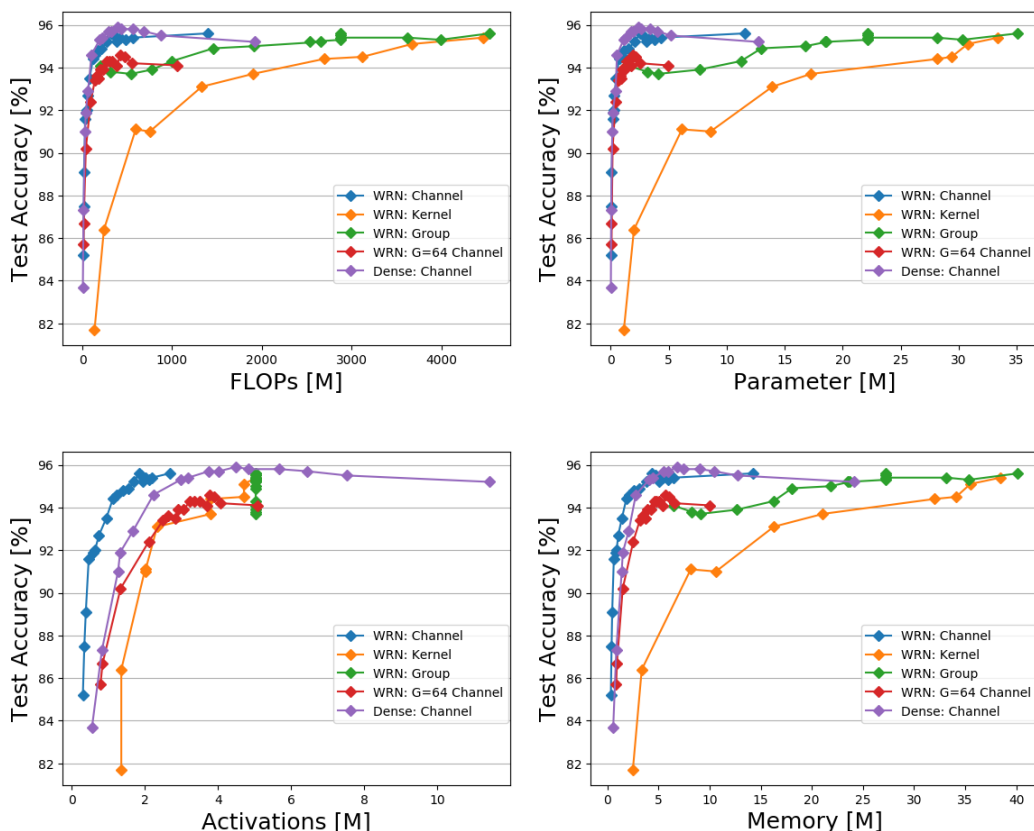
Figure 4: Performance metrics of several pruning structures and DNN architectures on CIFAR-10 for WRNs and DenseNets.

learn the number of input and output feature maps, kernel pruning to learn the size of the convolution kernel, and group pruning to learn heterogeneous group sizes for grouped convolutions (see Section 3.3.4 for a discussion on grouped convolutions).

In a first step, channel (*WRN: Channel*), kernel (*WRN: Kernel*), and group pruning (*WRN: Group*) are evaluated separately on the WRN architecture. The results for number of floating-point operations (FLOPs), parameters, activations, and memory (= parameters + activations) are reported in Figure 4. When considering the number of FLOPs and parameters, which are the main metrics in the literature on resource-efficient DNNs, it is clear that channel and group pruning significantly outperform kernel pruning. This indicates a high sensitivity of the kernel size to the overall accuracy. Group and channel pruning perform very similar with respect to FLOPs and parameters, especially for highly compressed models. However, when also considering the number of activations and overall memory consumption (i.e., parameters and activations), group pruning performs significantly worse since grouped convolutions only remove connections from input channels to output channels while keeping the overall number of channels the same. As a result, channel pruning is the best performing compression structure when applied in isolation.

In a next step, the group size is set to 64 and channel pruning is applied (*WRN: G=64 Channel*), in order to evaluate the performance when different sparse structures are combined. This combination performs worse than pure channel pruning for all metrics and requires a large amount of activations. Ultimately, it can be stated that group convolutions are excellent at reducing FLOPs and parameters but can harm the overall memory requirements by increasing the amount of activations.

Lastly, the DenseNet variant is compressed using channel pruning (*Dense: Channel*). The dense architecture outperforms the residual blocks in terms of number of FLOPs as well as parameters. In terms of number of activations, however, residual blocks are clearly more beneficial, which also influences the overall memory consumption. In summary, one can observe that DenseNets are more parameter/computation-efficient and ResNets are more memory-efficient.

5.2 Evaluating Compressed DNNs on Embedded Hardware

5.2.1 EVALUATING COMPRESSED DNNS ON CPU

Section 5.1 explored the impact of several network quantization approaches and structured pruning on the prediction quality. In this section. we use the well-performing LQ-Net approach for quantization and PSP (for channel pruning) to measure the inference throughput of the quantized and pruned models separately on an ARM Cortex-A53 CPU. The WRN model on the CIFAR-10 task is used again as a baseline, with a depth of 28 layers, varying widths of the model, and weights/activations quantized to different bit widths. Figure 5 reports test accuracies and throughput for different WRN variants and compression methods. Please note that multiple points for the same bit width correspond to a different width scaling of the model.

The results reveal that quantization does not provide throughput improvements on this processor. This is mainly due to the efficient floating-point units within the CPU in combination with fast on-chip memory and the high overhead resulting from performing low-bit-width computations. Moreover, the dimensions of some layers are too small to fit well to the bit level vectorized instructions and these layers limit the overall performance. Together with the accuracy reduction, low-bit-width quantization does not yield convincing results on this processor. Quantized DNNs with 1-bit weights and activations are the worst performing models, which is due to the severe implications of extreme quantization on prediction performance. As can be seen, however, the overall performance of the quantized models increases considerably when the bit width of activations is increased to 2–3 bit whilst the bit width of the weights is kept low. On the contrary, channel pruning consistently performs equally to the baseline model with respect to accuracy and throughput. Pruning is therefore the more suitable compression technique for this embedded processor, especially when considering that CPUs could potentially leverage a much finer sparsity granularity.

5.2.2 EVALUATING QUANTIZED DNNS ON FPGAS

While the previous section indicates that quantized DNNs do not provide throughput improvements on general-purpose processors without explicit hardware support, there are other hardware platforms where quantization is mandatory. Data-flow architectures, as found typically on FPGAs, where the main objective is to keep all required data for in-
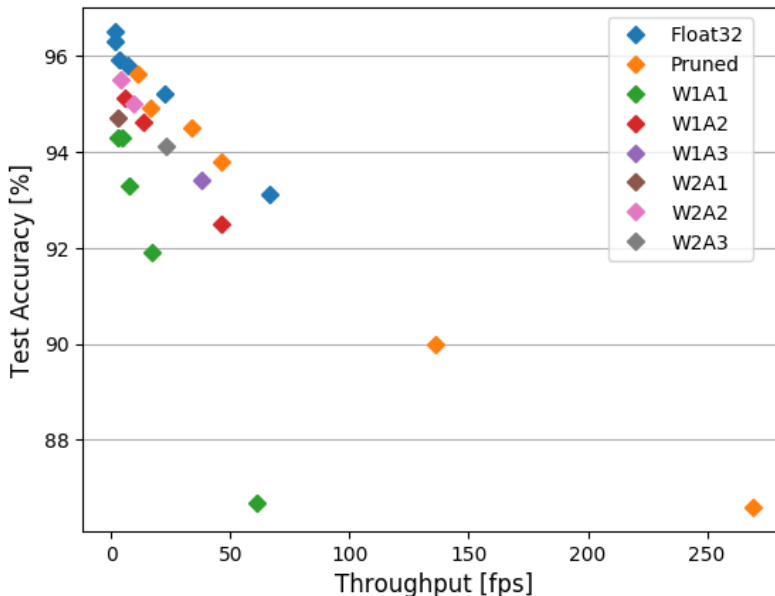
Figure 5: Throughput-accuracy trade-off of pruned and quantized WRN models on the CIFAR-10 task for an ARM CPU.

ference in on-chip memory, are usually constrained by the requirements for weight as well as activation storage. This section evaluates quantized DNNs on FPGAs using the FINN framework (Umuroglu et al., 2017) for generating data-flow architectures on reconfigurable hardware. Figure 6 shows test accuracy over throughput of the FINN data-flow architectures mapped to a XILINX Ultra96 FPGA using different bit combinations. A variant of the VGG architecture is used on the CIFAR-10 task for evaluation because FINN does not support residual connections yet, and the configuration of the FINN framework is adjusted so that highest throughput is targeted with respect to the available resources of the device (BRAM, LUTs, etc).

As expected, the test accuracy increases gradually with high bit widths while the throughput decreases accordingly. Following the Pareto front starting from the bottom right indicates that the best performing models use a combination of 1 bit for the weights and a gradual increase of activations up to 3 bits. Afterwards the models perform best if the weights are scaled to 2 bits and the activation bit width is further increased to 4 bits. This supports the observation of the previous sections, showing that model accuracy is sensitive to activation quantization rather than weight quantization.

5.2.3 Evaluating Pruned DNNs on GPU

Section 5.1.2 explored several pruning structures and DNN architectures which indicate different potential with respect to computation and memory efficiency. This section evaluates how these metrics impact the inference speed in terms of throughput. The same models are used as in Section 5.1.2, i.e., different WRN variants and DenseNet on the CIFAR-10
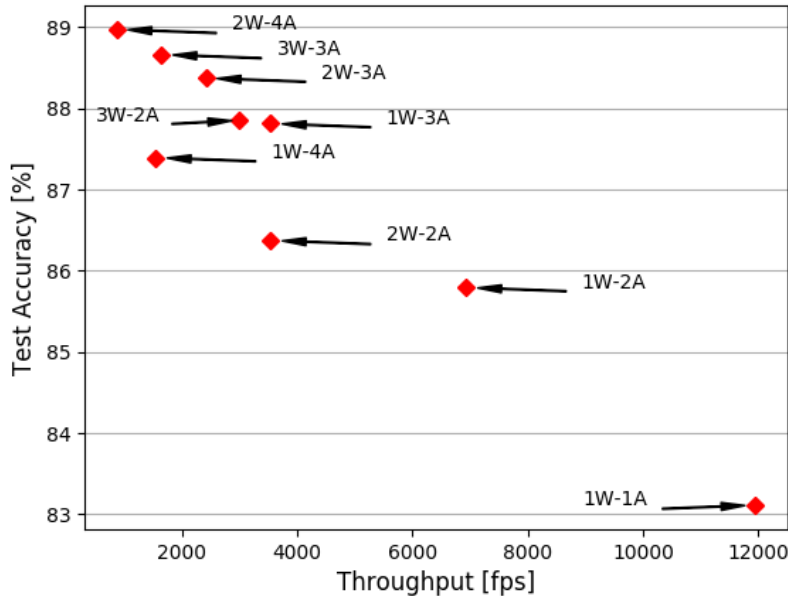
Figure 6: Throughput-accuracy trade-off of different quantized VGG models on the CIFAR-10 task for an FPGA data-flow architecture.

task. Figure 7 reports key inference metrics using the TensorRT framework targeting a Jetson Nano GPU. Half-precision floating-point is used for weights as well as activations and other reduced precision formats are omitted because the accelerator and its software infrastructure do not support arbitrary precision types.

As can be seen, the various models in each regime (pruning structure or DNN architecture) show similar behavior for throughput. The worst performing regimes are group and kernel pruning as well as the combination of fixed grouping and channel pruning. Especially interesting is group pruning: although it greatly reduces FLOPs and parameters, it fails at translating this reduction into faster computations. In contrast, employing pure channel pruning with residual and dense architectures yields the best performance. These results highlight the importance of reducing memory (or more specifically activations) rather than FLOPs in order to reduce latency or increase throughput.

5.2.4 OVERALL COMPARISON

The previous sections studied several compression methods with considerations on the targeted software and hardware stack. Each section had a focus on leveraging compression to accelerate inference computations as much as possible while maintaining the prediction quality of the uncompressed model. In this section we compare these specialized forms of compression on their respective hardware in terms of absolute performance to identify the most promising compute concepts for DNNs. Notably, whilst fundamentally different in architecture, from a system-level view these three processors, namely ARM Cortex-A57 CPU, NVIDIA Nano GPU, and XILINX Ultra96 FPGA, are comparable as they all exhibit
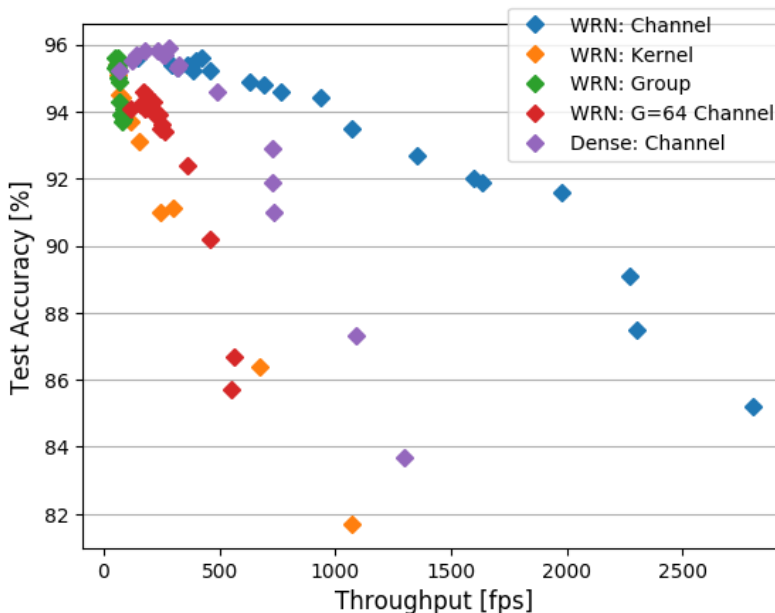
Figure 7: Throughput-accuracy trade-off of different pruning methods for an embedded GPU on the CIFAR-10 task.

a power consumption in the range of about 5 Watts. Figure 8 reports throughput and accuracy for these devices using different compression methods.

CPUs are well suited for mapping compressed DNNs. However, the comparison to massively parallel processors shows that they lack the necessary amount of parallelism to achieve competitive throughput. The benefit of CPUs is that they feature a relatively large memory, allowing large and accurate models to be deployed.

FPGAs excel at extremely-high-throughput inference and high utilization of the available hardware resources. Such data-flow architectures, however, demand the entire model (including activations) to stay on chip, possibly preventing larger and more accurate models to be deployed.

GPUs are relatively constrained in terms of flexibility when deploying compressed models, due to the requirement of using optimized libraries and their respective software stack. However, they show a good compromise of programmable, general-purpose processing, enabling high throughput and accuracy. Additionally, they feature a large off-chip memory which in combination with latency hiding techniques enables high-throughput inference for large models.

6. Conclusion

We presented an overview of the vast literature of the highly active research area concerned with resource efficiency of DNN inference. We have identified three major directions of research, namely (i) network quantization, (ii) network pruning, and (iii) approaches that
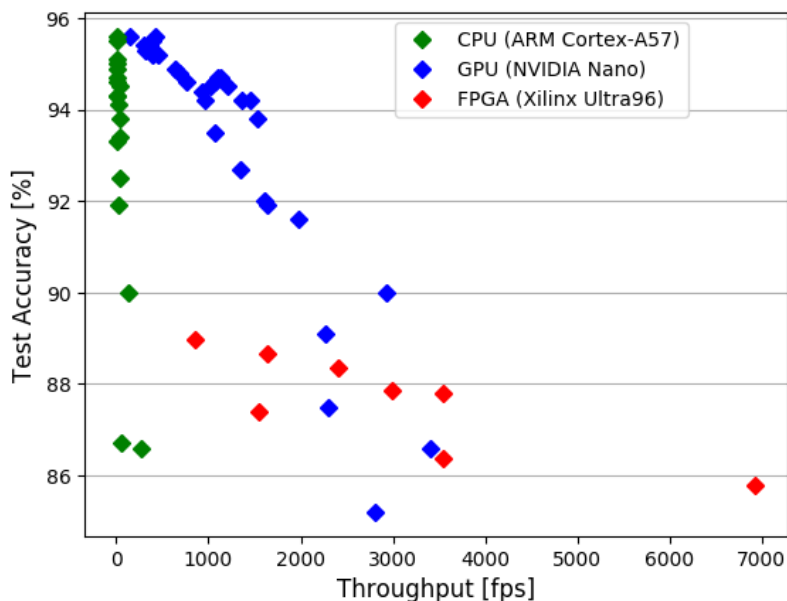
Figure 8: Throughput-accuracy trade-off of different compression methods for different processor architectures (CPU, FPGA, GPU) on the CIFAR-10 task.

target efficiency at the structural level. Many of the presented works are orthogonal and can in principle be used in combination to potentially further improve the results reported in the respective papers.

We have discovered several patterns in the individual strategies for enhancing resource efficiency. For quantization approaches, a common pattern in the most successful approaches is to combine real-valued representations, that help in maintaining the expressiveness of DNNs, with quantization to enhance the computationally intensive operations. More recently, mixed-precision quantization, where the bit widths are determined during training, is an upcoming topic. For pruning methods, we observed that the trend is moving towards structured pruning approaches that obtain smaller models whose data structures are compatible with highly optimized dense tensor operations. On the structural level of DNNs, a lot of progress has been made in the development of specific architectures that maintain a high expressiveness of the DNN while at the same time reducing the computational overhead substantially. The newly emerging NAS approaches are promising candidates to automate the design of application-specific architectures with little user interaction. However, it appears unlikely that current NAS approaches will discover new fundamental design principles as the resulting architectures highly depend on a-priori knowledge encoded in the architecture search space.

In experiments, we demonstrated on two benchmark data sets the difficulty of finding a good trade-off among prediction quality, representational efficiency and computational efficiency. Considering three embedded hardware platforms, we showed that massive parallelism is required for inference efficiency and that quantization as well as structured pruning map well onto these accelerators. We furthermore point out that hardware properties and

the corresponding computational efficiency form a large fraction of resource efficiency. This highlights the need to consider particular hardware targets when searching for resource-efficient machine learning models.

Acknowledgments

References

Jan Achterhold, Jan M. Köhler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations (ICLR)*, 2018.

Alexander G. Anderson and Cory P. Berg. The high-dimensional geometry of binary neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems (NIPS)*, pages 2654–2662, 2014.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning (ICML)*, pages 1613–1622, 2015.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 535–541, 2006.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave Gaussian quantization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5406–5414, 2017.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pages 2285–2294, 2015.

Yu Cheng, Felix X. Yu, Rogério Schmidt Feris, Sanjiv Kumar, Alok N. Choudhary, and Shih-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *International Conference on Computer Vision (ICCV)*, pages 2857–2865, 2015.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research (JMLR)*, 24(240):1–113, 2023.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. In *International Conference on Learning Representations (ICLR) Workshop*, volume abs/1412.7024, 2015a.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3123–3131, 2015b.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013.

R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 1974.

Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1269–1277, 2014.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1895–1903, 2017.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *International Conference on Computer Vision (ICCV)*, pages 293–302, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, Jun 2021.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research (JMLR)*, 20(1):1997–2017, 2019.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Advait Harshal Gadhikar, Sohom Mukherjee, and Rebekka Burkholz. Why random pruning is all we need to start sparse. In *Proceedings of Machine Learning Research (PMLR)*, volume 202, 2023.

Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert Mullins, and Cheng-Zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations (ICLR)*, 2019.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2348–2356, 2011.

Peter D. Grünwald. *The minimum description length principle*. MIT press, 2007.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1379–1387, 2016.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, pages 1737–1746, 2015.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS)*, pages 164–171, 1992.

Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *International Conference on Learning Representations (ICLR)*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Conference on Computational Learning Theory (COLT)*, pages 5–13, 1993.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop @ NIPS*, 2015.

Markus Höhfeld and Scott E. Fahlman. Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Transactions on Neural Networks*, 3(4):602–611, 1992a.

Markus Höhfeld and Scott E. Fahlman. Probabilistic rounding in neural network learning with limited precision. *Neurocomputing*, 4(6):291–299, 1992b.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, May 2019. URL `https://arxiv.org/abs/1905.02244`.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017a.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, Apr 2017b. URL `https://arxiv.org/abs/1704.04861`.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *European Conference on Computer Vision (ECCV)*, pages 317–334, 2018.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4107–4115, 2016.

Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Prabhu Kaliamoorthi, Aditya Siddhant, Edward Li, and Melvin Johnson. Distilling large language models into tiny and effective students using pQRNN. *CoRR*, abs/2101.08890, 2021.

Jangho Kim, Seonguk Park, and Nojun Kwak. Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2765–2774, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2575–2583, 2015.

Anoop Korattikara, Vivek Rathod, Kevin P. Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3438–3446, 2015.

Torben Krieger, Bernhard Klein, and Holger Fröning. Towards hardware-specific automatic compression of neural networks. *CoRR*, abs/2212.07818, Dec 2022. URL https://arxiv.org/abs/2212.07818.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*, pages 598–605, 1989.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5811–5821, 2017.

Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. Learning small-size DNN with output-distribution-based criteria. In *INTERSPEECH: Conference of the International Speech Communication Association*, pages 1910–1914, 2014.

Yang Li and Shihao Ji. L_0-ARM: Network sparsification via stochastic binary optimization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019.

Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning (ICML)*, pages 2849–2858, 2016.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2181–2191, 2017a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *CoRR*, abs/2306.00978, 2023.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014a.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014b.

Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Neural Information Processing Systems (NIPS)*, pages 345–353, 2017b.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. In *International Conference on Learning Representations (ICLR)*, 2015.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019a.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *European Conference on Computer Vision (ECCV)*, pages 747–763, 2018.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*, 2019b.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3288–3298, 2017.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations (ICLR)*, 2018.

Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *International Conference on Computer Vision (ICCV)*, pages 5068–5076, 2017.

Zelda Mariet and Suvrit Sra. Diversity networks: Neural network compression using determinantal point processes. In *International Conference on Learning Representations (ICLR)*, 2016.

Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12), 2023.

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence (UAI)*, pages 362–369, 2001.

Asit K. Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *International Conference on Learning Representations (ICLR)*, 2018.

Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *CoRR*, abs/1603.01025, 2016.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, pages 2498–2507, 2017.

Radford M. Neal. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Technical report, Dept. of Computer Science, University of Toronto, 1992.

Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer New York, 2 edition, 2006.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 442–450, 2015.

Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

Jorn W. T. Peters and Max Welling. Probabilistic binary neural networks. *CoRR*, abs/1809.03368, 2018.

Mary Phuong and Christoph Lampert. Distillation-based training for multi-exit architectures. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1355–1364, 2019.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations (ICLR)*, 2018.

Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 814–822, 2014.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pages 525–542, 2016.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

Wolfgang Roth and Franz Pernkopf. Bayesian neural networks with weight sharing using Dirichlet processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018.

Wolfgang Roth, Günther Schindler, Holger Fröning, and Franz Pernkopf. Training discrete-valued neural networks with sign activations using weight distributions. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, Jan 2018a. URL `https://arxiv.org/abs/1801.04381`.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018b.

Günther Schindler, Wolfgang Roth, Franz Pernkopf, and Holger Fröning. Parameterized structured pruning for deep neural networks. In *6th International Conference on Machine Learning, Optimization, and Data Science (LOD)*, 2020.

Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. In *International Conference on Learning Representations (ICLR)*, 2018.

Alexander Shekhovtsov, Viktor Yanush, and Boris Flach. Path sample-analytic gradient estimators for stochastic binary networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 12884–12894, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL `https://arxiv.org/abs/1909.08053`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems (NIPS)*, pages 963–971, 2014.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: Designing hardware-efficient convnets in less than 4 hours. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, May 2019. URL `https://arxiv.org/abs/1905.11946`.

Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, Apr 2021.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6), 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, 2023. URL `https://arxiv.org/abs/2302.13971`.

Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso García, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision DNNs: All you need is a good parametrization. In *International Conference on Learning Representations (ICLR)*, 2020.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations (ICLR)*, 2017.

Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA)*, pages 65–74, 2017.

Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5741–5752, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, Jun 2017. URL `https://arxiv.org/abs/1706.03762`.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8612–8620, 2019.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning (ICML)*, pages 681–688, 2011.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2074–2082, 2016.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *CoRR*, abs/1812.00090, 2018a. URL http://arxiv.org/abs/1812.00090.

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2018b.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alexander J. Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *International Conference on Computer Vision (ICCV)*, pages 1476–1483, 2015.

Mingzhang Yin and Mingyuan Zhou. ARM: augment-REINFORCE-merge gradient for stochastic binary networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

Xinchuan Zeng and Tony R. Martinez. Using a neural network to approximate an ensemble of classifiers. *Neural Processing Letters*, 12(3):225–237, 2000.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *European Conference on Computer Vision (ECCV)*, pages 373–390, 2018a.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018b.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018.