An Overview of Neural Network Compression

James T. O' Neill
Department of Computer Science
University of Liverpool
Liverpool, England, L69 3BX
james.o-neill@liverpool.ac.uk

Abstract

Overparameterized networks trained to convergence have shown impressive performance in domains such as computer vision and natural language processing. Pushing state of the art on salient tasks within these domains corresponds to these models becoming larger and more difficult for machine learning practitioners to use given the increasing memory and storage requirements, not to mention the larger carbon footprint. Thus, in recent years there has been a resurgence in model compression techniques, particularly for deep convolutional neural networks and self-attention based networks such as the Transformer.

Hence, this paper provides a timely overview of both old and current compression techniques for deep neural networks, including pruning, quantization, tensor decomposition, knowledge distillation and combinations thereof.

We assume a basic familiarity with deep learning architectures[1], namely, Recurrent Neural Networks [(RNNs) 155, 75], Convolutional Neural Networks [50] [2] and Self-Attention based networks [181][3],[4]. Most of the papers discussed are proposed in the context of at least one of these DNN architectures.

[1]For an introduction to deep learning, see Goodfellow et al. [54]

[2]For an up to date overview see Khan et al. [92]

[3]For a general overview of self-attention networks, see Chaudhari et al. [22].

[4]For more detail and their use in natural language processing, see Hu [79]

Preprint. Under review.

Contents

1 Introduction

Deep neural networks (DNN) are becoming increasingly large, pushing the limits of generalization performance and tackling more complex problems in areas such as computer vision (CV), natural language processing (NLP), robotics and speech to name a few. For example, Transformer-based architectures [181, 158, 117, 196, 100, 41] that are commonly used in NLP (also used in CV to a less extent [144]) have millions of parameters for each fully-connected layer. Tangentially, Convolutional Neural Network [(CNN) 50] based architectures [97, 67, 203, 66] used in vision and NLP tasks Kim [93], Hu et al. [78], Gehring et al. [51]).

From the left of Figure 1, we see that in general, larger overparameterized CNN networks generalize better for ImageNet (a large image classification benchmark dataset). However, recent architectures that aim to reduce the number of floating point operations (FLOPs) and improve training efficiency with less parameters have also shown impressive performance e.g EfficientNet [173].

The increase in Transformer network size, shown on the right, is more pronounced given that the network consists of fully-connected layers that contain many parameters in each self-attention block [181]. MegatronLM [166], shown on the right-hand side, is a 72-layer GPT-2 model consisting of 8.3 billion parameters, trained by using 8-way model parallelism and 64-way data parallelism over 512 GPUs. Rosset [154] proposed a 17 billion parameter Transformer model for natural language text generation (NLG) that consists of 78 layers with hidden size of 4,256 and each block containing 28 attention heads. They use *DeepSpeed*[5] with ZeRO [148] to eliminate memory redundancies in data parallelism and model parallelism and allow for larger batch sizes (e.g 512), resulting in three times faster training and less GPUs required in the cluster (256 instead of 1024). Brown et al. [14] the most recent Transformer to date, trains a GPT-3 autoregressive language model that contains 175 billion parameters. This model can perform NLP tasks (e.g machine translation, question-answering) and digit arithmetic relatively well with only few examples, closing the performance gap to similarly large pretrained models that are further fine-tuned for specific tasks and in some cases outperforming them given the large increase in the number of parameters. The resources required to store the aforementioned CNN and Transformer models on a/many GPU/s let alone train one is out of reach for a large majority of machine learning practitioners. Moreover, these models have predominantly been driven by improving the state of the art (SoTA) and pushing the boundaries of what complex tasks can be solved using them. Therefore, we expect that the current trend of increasing network size will remain.

Thus, the motivation to compress models has grown and expanded in recent years from being predominantly focused around deployment on mobile devices, to also learning smaller networks on the same device but with eased hardware constraints i.e learning on a small number of Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) or the same number of GPUs and TPUs but with a smaller amount of VRAM. For these reasons, model compression can be viewed as

[5] A library that allows for distributed training with mixed precision (MP), model parallelism, memory optimization, clever gradient accumulation, loss scaling with MP, large batch training with specialized optimizers, adaptive learning rates and advanced parameter search. See here `https://github.com/microsoft/DeepSpeed.git`

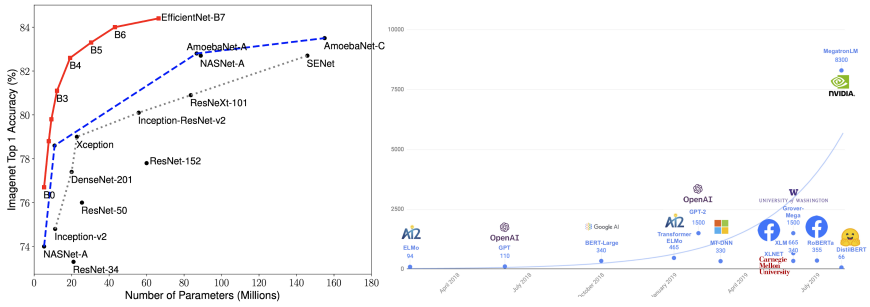


Figure 1: Accuracy vs # Parameters for CNN architectures (source on left: Tan and Le [172]) and # Parameters vs Years for Transformers (source on right: Sanh [157])

a critical research endeavour to allow the machine learning community to continue to deploy and understand these large models with limited resources.

Hence, this paper provides an overview of methods and techniques for compressing DNNs. This includes weight sharing (section 2), pruning (section 3), tensor decomposition (section 4), knowledge distillation (section 5) and quantization (section 6). Retraining is often required to account for some performance loss in each case. The retraining step can be carried out using unsupervised (including self-supervised) learning (e.g tensor decomposition) and supervised learning (knowledge distillation (knowledge distillation). Unsupervised compression is often used when there is are no particular target task/s that the model is being specifically compressed for, alternatively supervision can be used to gear the compressed model towards a subset of tasks in which case the target task labels are used as opposed to the original data the model was trained on, unlike unsupervised model comrpression. In some cases, RLg has also shown to be beneficial for maintaining performance during iterative pruning [111], knowledge distillation [6] and quantization [197].

Further Motivation for Compression A question that may naturally arise at this point is - *Can we obtain the same or similar generalization performance by training a smaller network from scratch to avoid training a larger teacher network to begin with ?*

Before the age of DNNs, Castellano et al. [21] found that training a smaller shallow network from random initialization has poorer generalization compared to an equivalently sized pruned network from a larger pretrained network.

More recently, Zhu and Gupta [211] have too addressed this question, specifically in the case of pruning DNNs - that is, whether many previous work that previously report large reductions in pretrained networks using pruning were already severely overparameterized and could be achieved by simply training a smaller model equivalent in size to the pruned network. They find for deep CNNs and LSTMs that large sparsely pruned networks consistently outperform smaller dense models, achieving a compression ratio of 10 in the number of non-zero parameters with minuscule losses in accuracy. Even when the pruned network is not necessarily overparameterized in the pretraining stage, it still produces consistently better performance than an equivalently sized network trained from scratch [111].

Essentially, having a DNN with larger capacity (i.e more degrees of freedom) allows for a larger set of solutions to be chosen from in the parameter space. Overparameterization has also shown to have a smoothening effect on the loss space [108] when trained with stochastic gradient descent (SGD) [44, 30], in turn producing models that generalize better than smaller models. This has been reinforced recently for DNNs after the discovery of the double descent phenomena [12], whereby a 2^{nd} descent in the test error is found for overparameterized DNNs that have little to no training errors, occurring after the *(*critical regime) region where the test error is initially high. This 2^{nd} descent in test error tends to converge to an error lower than that found in the 1^{st} descent where the 1^{st} descent corresponds to the traditional bias-variance tradeoff. Moreover, the norm of the weights becomes dramatically smaller in each layer during this 2^{nd} descent, during the compression phase [167]. Since the weights tend to be close to zero when trained far into this 2^{nd} region, it becomes clearer why compression techniques, such as pruning, has less effect on the networks behaviour when trained to convergence since the magnitude of individual weights becomes smaller as the network grows larger.

Frankle and Carbin [48] also showed that training a network to convergence with more parameters makes it easier to find a subnetwork that when trained from scratch, maintains performance, further suggesting that compressing large pretrained overparameterized DNNs that are trained to convergence has advantages from performance and storage perspective over training and equivalently smaller DNN. Even in cases when the initial compression causes a degradation in performance, retraining the compressed model can and is commonly carried out to maintain performance.

Lastly, large pretrained models are widely and publicly available[6][7] and thus can be easily used and compared by the rest of the machine learning community, avoiding the need to train these models from scratch. This further motivates the utility of model compression and its advantages over training equivalently smaller network from scratch.

[6]pretrained CNN models:`https://github.com/Cadene/pretrained-models.pytorch`
[7]pretrained Transformer models:`https://github.com/huggingface/transformers`

Compression Evaluation Metrics Lastly, we note that the main evaluation of compression techniques is performance metric (e.g accuracy) vs model size. When evaluating for speedups obtained from the model compression, the number of floating point operations (FLOPs) is a commonly used metric. When claims of storage improvements are made, this can be demonstrated by reporting the run-time memory footprint which is essentially the ratio of the space for storing hidden layer features during run time when compared to the original network.

We now begin to describe work for each compression type, beginning with weight sharing.

2 Weight Sharing

The simplest form of network reduction involves sharing weights between layers or structures within layers (e.g filters in CNNs). We note that unlike compression techniques discussed in later sections (Section 3-6), standard weight sharing is carried out prior to training the original networks as opposed to compressing the model after training. However,recent work which we discuss here [24, 180, 9] have also been used to reduce DNNs post-training and hence we devote this section to this straightforward and commonly used technique.

Weight sharing reduces the network size and avoids sparsity. It is not always clear how many and what group of weights should be shared before there is an unacceptable performance degradation for a given network architecture and task. For example, Inan et al. [83] find that tying the input and output representations of words leads to good performance while dramatically reducing the number of parameters proportional to the size of the vocabulary of given text corpus. Although, this may be specific to language modelling, since the output classes are a direct function of the inputs which are typically very high dimensional (e.g typically greater than 10^6). Moreover, this approach assigns the embedding matrix to be shared, as opposed to sharing individual or sub-blocks of the matrix. Other approaches include clustering weights such that their centroid is shared among each cluster and using weight penalty term in the objective to group weights in a way that makes them more amenable to weight sharing. We discuss these approaches below along with other recent techniques that have shown promising results when used in DNNs.

2.1 Clustering-based Weight Sharing

Nowlan and Hinton [137] instead propose a soft weight sharing scheme by learning a Gaussian Mixture Model that assigns groups of weights to a shared value given by the mixture model. By using a mixture of Gaussians, weights with high magnitudes that are centered around a broad Gaussian component are under less pressure and thus penalized less. In other words, a Gaussian that is assigned for a subset of parameters will force those weights together with lower variance and therefore assign higher probability density to each parameter.

Equation 1 shows the cost function for the Gaussian mixture model where $p(w_j)$ is the probability density of a Gaussian component with mean μ_j and standard deviation σ_j. Gradient Descent is used to optimize w_i and mixture parameters π_j, μ_j, σ_j and σ_y.

$$C = \frac{K}{\sigma_y^2} \sum_c (y_c - d_c)^2 - \sum_i \log \Big[\sum_j \pi_j p_j(w_i) \Big] \tag{1}$$

The expectation maximization (EM) algorithm is used to optimize these mixture parameters. The number of parameters tied is then proportional to the number of mixture components that are used in the Gaussian model.

An Extension of Soft-Weight Sharing Ullrich et al. [180] build on soft-weight sharing [137] with factorized posteriors by optimizing the objective in Equation 2. Here, $\tau = 5e^{-3}$ controls the influence of the log-prior means μ, variances σ and mixture coefficients π, which are learned during retraining apart from the j-th component that are set to $\mu_j = 0$ and $\pi_j = 0.99$. Each mixture parameter has a learning rate set to 5×10^{-4}. Given the sensitivity of the mixtures to collapsing if the correct

hyperparameters are not chosen, they also consider the inverse-gamma hyperprior for the mixture variances that is more stable during training.

$$\mathcal{L}\big(w, \{\mu_j, \sigma_j, \pi_j\}_{j=0}^{J}\big) = \mathcal{L}_E + \tau\mathcal{L}_C = -\log p\big(\tau|X, w\big) - \tau \log p\big(w, \{\mu_j, \sigma_j, \pi_j\}_{j=0}^{J}\big) \tag{2}$$

After training with the above objective, if the components have a KL-divergence under a set threshold, some of these components are merged [1] as shown in Equation 3. Each weight is set then set to the mean of the component with the highest mixture value $\mathrm{argmax}(\pi)$, performing GMM-based quantization.

$$\pi_{\text{new}} = \pi_i + \pi_j, \quad \mu_{\text{new}} = \frac{\pi_i\mu_i + \pi_j\mu_j}{\pi_i + \pi_j}, \quad \sigma^2_{\text{new}} = \frac{\pi_i\sigma_i^2 + \pi_j\sigma_j^2}{\pi_i + \pi_j} \tag{3}$$

In their experiments, 17 Gaussian components were merge to 6 quantization components, while still leading to performance close to the original LeNet classifier used on MNIST.

2.2 Sharing via Weight Regularization

Learning Weights Sharing Zhang et al. [204] explicitly try to learn which weights should be shared by imposing a group order weighted ℓ_1 (GrOWL) sparsity regularization term while simultaneously learning to group weights and assign them a shared value. In a given compression step, groups of parameters are identified for weight sharing using the aforementioned sparsity constraint and then the DNN is retrained to fine-tune the structure found via weight sharing. GrOWL first identify the most important weights and then clusters correlated features to learn the values of the closest important weight throughout training. This can be considered an adaptive weight sharing technique.

Parameter Hashing Chen et al. [24] use hash functions to randomly group weight connections into hash buckets that all share the same weight value. Parameter hashing [189, 165] can easily be used with backpropogation whereby each bucket parameters have subsets of weights that are randomly i.e each weight matrix contains multiple weights of the same value (referred to as a *virtual matrix*), unlike standard weight sharing where all weights in a matrix are shared between layers.

2.3 Weight Sharing in Large Architectures

Weight Sharing in Transformers Dehghani et al. [38] propose Universal Transformers (UT) to combine the benefits of recurrent neural networks (recurrent inductive bias) with Transformers [181] (parallelizable self-attention and its global receptive field). As apart of UT, weight sharing to reduce the network size showed strong results on NLP defacto benchmarks while .

Dabre and Fujita [33] use a 6-hidden layer Transformer network for neural machine translation (NMT) where the same weights are fed back into the same attention block recurrently. This straightforward approach surprisingly showed similar performance of an untied 6-hidden layer for standard NMT benchmark datasets.

Xiao et al. [192] use shared attention weights in Transformer as dot-product attention can be slow during the auto-regressive decoding stage. Attention weights from hidden states are shared among adjacent layers, drastically reducing the number of parameters proportional to number of attention heads used. The Jenson-Shannon (JS) divergence is taken between self-attention weights of different heads and they average them to compute the average JS score. They find that the weight distribution is similar for layers 2-6 but larger variance is found among encoder-decoder attention although some adjacent layers still exhibit relatively JS score. Weight matrices are shared based on the JS score whereby layers that have JS score larger than a learned threshold (dynamically updated throughout training) are shared. The criterion used involves finding the largest group of attention blocks that have similarity above the learned threshold to maximize largest number of weight groups that can be shared while maintaining performance. They find a 16 time storage reduction over the original Transformer while maintaining competitive performance.

Deep Equilibrium Model Bai et al. [9] propose deep equilibrium models (DEMs) that use a root-finding method to find the equilibrium point of a network and can be analytically backpropogated through at the equilibrium point using implicit differentiation. This is motivated by the observation that hidden states of sequential models converge towards a fixed point. Regardless of the network depth, the approach only requires constant memory because backpropogration only needs to be performed on the layer of the equilibrium point.

For a recurrent network $f_{\mathbf{W}}(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})$ of infinite hidden layer depth that takes inputs $\boldsymbol{x}_{1:T}$ and hidden states $\boldsymbol{z}_{1:T}$ up to T timesteps, the transformations can be expressed as,

$$\lim_{i\to\infty} z^{[i]}_{1:T} = \lim_{i\to\infty} f_W(\boldsymbol{Z}^{[i]}_{1:T}; \boldsymbol{x}_{1:T}) := f_W(\boldsymbol{z}^*_{1:T}; x_{1:T}) = \underbrace{\boldsymbol{z}^*_{1:T}}_{\text{equilibrium point}} \tag{4}$$

where the final representation $\boldsymbol{z}^*_{1:T}$ is the hidden state output corresponds to the equilibrium point of the network. They assume that this equilibrium point exists for large models, such as Transformer and Trellis [8] networks (CNN-based architecture).

The $\frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}}$ requires implicit differentiation and Equation 5 can be rewritten as Equation 6.

$$\frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}} = \frac{df_W(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{d\mathbf{W}} + \frac{\partial f_W(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{\partial \boldsymbol{z}^*_{1:T}} \frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}} \tag{5}$$

$$\Big(I - \frac{\partial f_W(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{\partial \boldsymbol{z}^*_{1:T}}\Big) \frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}} = \frac{df_W(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{d\,\mathbf{W}} \tag{6}$$

For notational convenience they define $g_{\mathbf{W}}(z^*_{1:T}; \boldsymbol{x}_{1:T}) = f_{\mathbf{W}}(z^*_{1:T}; \boldsymbol{x}_{1:T}) - \boldsymbol{z}^*_{1:T} \to 0$ and thus the equilibrium state $\boldsymbol{z}^*_{1:T}$ is thus the root of $g_{\mathbf{W}}$ found by the Broyden's method [15][8].

The Jacobian of the function $g_{\mathbf{W}}$ at the equilibrium point $\boldsymbol{z}^*_{1:T}$ w.r.t $\mathbf{W}$ can then be expressed as Equation 7. Note that this is computed without having to consider how the equilibrium $\boldsymbol{z}^*_{1:T}$ was obtained.

$$\mathbf{J}_{g_{\mathbf{W}}}\Big|_{\boldsymbol{z}^*_{1:T}} = -\Big(I - \frac{\partial f_W(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{\partial \boldsymbol{z}^*_{1:T}}\Big) \tag{7}$$

Since $f_{\mathbf{W}}(\cdot)$ is in equilibrium at $\boldsymbol{z}^*_{1:T}$ they do not require to backpropogate through all the layers, assuming all layers are the same (this is why it is considered a weight sharing technique). They only need to solve Equation 8 to find the equilibrium points using Broydens method,

$$\frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}} = -\mathbf{J}_{g_{\mathbf{W}}}\Big|_{\boldsymbol{z}^*_{1:T}} \frac{d\, f_{\mathbf{W}}(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{d\mathbf{W}} \tag{8}$$

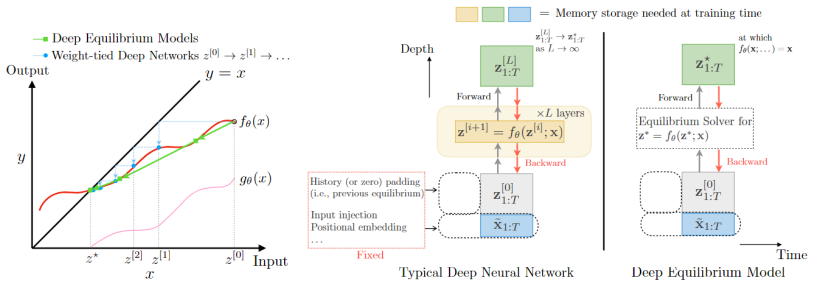
and then perform a single layer update using backpropogation at the equilibrium point.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\mathcal{L}}{\partial \boldsymbol{z}^*_{1:T}} \frac{\partial \boldsymbol{z}^*_{1:T}}{\partial \mathbf{W}} = -\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^*_{1:T} \Big(\mathbf{J}^{-1}_{g_{\mathbf{W}}}\Big|_{\boldsymbol{z}^*_{1:T}}\Big)} \frac{d\, f_{\mathbf{W}}(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{d\mathbf{W}} \tag{9}$$

The benefit of using Broyden method is that the full Jacobian does not need to be stored but instead an approximation $\hat{\mathbf{J}}^{-1}$ using the Sherman-Morrison formula [161] which can then be used as apart of the Broyden iteration:

$$\boldsymbol{z}^{[i+1]}_{1:T} := \boldsymbol{z}^{[i]}_{1:T} - \alpha \hat{\mathbf{J}}^{-1}_{g_{\mathbf{W}}}\Big|_{\boldsymbol{z}^{[i]}_{1:T}} g_{\mathbf{W}}(\boldsymbol{z}^{[i]}_{1:T}; \boldsymbol{x}_{1:T}) \quad \text{for} \quad i = 0, 1, 2, \ldots \tag{10}$$

[8] a quasi-Newton method for finding roots of a parametric model



(a) A simple illustration of solving for an equilibrium point in 2D.

(b) A deep equilibrium model operates with significantly less memory than conventional deep nets due to an analytical backward pass.

Figure 2: original source Bai et al. [9]: Comparison of the DEQ with conventional weight-tied deep networks

where $alpha$ is the learning rate. This update can then be expressed as Equation 11

$$\mathbf{W}^+ = \mathbf{W} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{W} + \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{z} *_{1:T} \left(\mathbf{J}_{g_\mathbf{W}}^{-1} \big|_{\boldsymbol{z}^*_{1:T}} \right)} \frac{d\, f_\mathbf{W}(\boldsymbol{z}^*_{1:T}; \boldsymbol{x}_{1:T})}{d\mathbf{W}} \tag{11}$$

Figure 2 shows the difference between a standard Transformer network forward pass and backward pass in comparison to DEM passes. The left figure illustrates the Broyden iterations to find the equilibrium point for inputs over successive inputs. On WikiText-103, they show that DEMs can improve SoTA sequence models and reduce memory by 88% use for similar computational requirements as the original models.

3 Network Pruning

Pruning weights is perhaps the most commonly used technique to reduce the number of parameters in a pretrained DNN. Pruning can lead to a reduction of storage and model runtime and performance is usually maintaining by retraining the pruned network. Iterative weight pruning prunes while retraining until the desired network size and accuracy tradeoff is met. From a neuroscience perspective, it has been found that as humans learn they also carry out a similar kind of iterative pruning, removing irrelevant or unimportant information from past experiences [182]. Similarly, pruning is not carried out at random, but selected so that unimportant information about past experiences is discarded. In the context of DNNs, random pruning (akin to Binary Dropout) can be detrimental to the models performance and may require even more retraining steps to account for the removal of important weights or neurons [200].

The simplest pruning strategy involves setting a threshold γ that decides which weights or units (in this case, the absolute sum of magnitudes of incoming weights) are removed [59]. The threshold can be set based on each layers weight magnitude distribution, where weights centered around the mean μ are removed, or it the threshold can be set globally for the whole network. Alternatively, pruning the weights with lowest absolute value of the normalized gradient multiplied by the weight magnitude [104] for a given set of mini-batch inputs can be used, either layer-wise or globally too.

Instead of setting a threshold, one can predefine a percentage of weights to be pruned based on the magnitude of w, or a percentage aggregated by weights for each layer w_l, $\forall l \in L$. Most commonly, the percentage of weights that are closest to 0 are removed. The aforementioned criteria for pruning are all types of *magnitude-based pruning* (MBP). MBP has also been combined with other strategies such as adding new neurons during iterative pruning to further improve performance [62, 134], where the number of new neurons added is less than the number pruned in the previous pruning step and so the overall number of parameters monotonically decreases.

MBP is the most commonly used in DNNs due to its simplicity and performs well for a wide class of machine learning models (including DNNs) on a diverse range of tasks [163]. In general, global MBP tends to outperform layer-wise MBP [90, 151, 59, 104], because there is more flexibility on the amount of sparsity for each layer, allowing more salient layer to be more dense while less salient

to contain more non-zero entries. Before discussing more involved pruning methods, we first make some important categorical distinctions.

3.1 Categorizing Pruning Techniques

Pruning algorithms can be categorized into those that carry out pruning without retraining the pruning and those that do. Retraining is often required when pruning degrades performance. This can happen when the DNN is not necessarily overparameterized, in which case almost all parameters are necessary to maintain good generalization.

Pruning techniques can also be categorized into what type of criteria is used as follows:

1. The aforementioned magnitude-based pruning whereby the weights with the lowest absolute value of the weight are removed based on a set threshold or percentage, layer-wise or globally.
2. Methods that penalize the objective with a regularization term to force the model to learn a network with (e.g ℓ_1, ℓ_2 or lasso weight regularization) smaller weights and prune the smallest weights.
3. Methods that compute the sensitivity of the loss function when weights are removed and using this as a criterion for removing weights that result in the smallest change in loss.
4. Search-based approaches (e.g particle filters, evolutionary algorithms, reinforcement learning) that seek to learn or adapt a set of weights to links or paths within the neural network and keep those which are salient for the task. Unlike (1) and (2), the pruning technique does not involve gradient descent as apart of the pruning criteria (with the exception of using deep RL).

Unstructured vs Structured Pruning Another important distinction to be made is that between structured and unstructured pruning techniques where the latter aims to preserve network density for computational efficiency (faster computation at the expense of less flexibility) by removing groups of weights, whereas unstructured is unconstrained to which weights or activations are removed but the sparsity means that the dimensionality of the layers does not change. Hence, sparsity in unstructured pruning techniques provide good performance at the expense of slower computation. For example, MBP produces a sparse network that requires sparse matrix multiplication (SMP) libraries to take full advantage of the memory reduction and speed benefits for inference. However, SMP is generally slower than dense matrix multiplication and therefore there has been work towards preserving subnetworks which omit the need for SMP libraries (discussed in subsection 3.4).

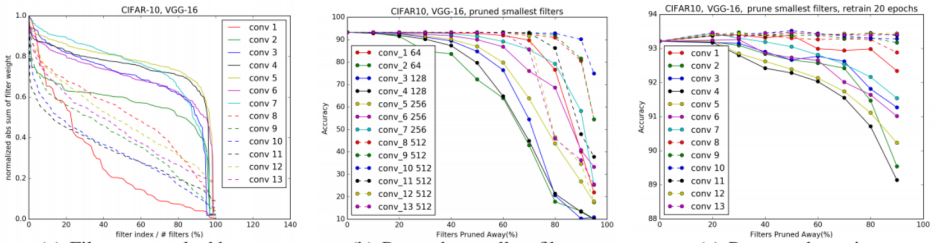
With these categorical distinctions we now move on to the following subsections that describe various pruning approaches beginning with pruning by using weight regularization.

3.2 Pruning using Weight Regularization

Constraining the weights to be close to 0 in the objective function by adding a penalty term and deleting the weights closest to 0 post-training can be a straightforward yet effective pruning approach. Equation 12 shows the commonly used ℓ_2 penalty that penalizes large weights w_m in the m-th hidden layer with a large magnitude and $\boldsymbol{v}_m$ are the output layer weights of output dimension C.

$$C(\boldsymbol{w}, \boldsymbol{v}) = \frac{\epsilon}{2}\Big(\sum_{m=1}^{h} \sum_{l=1}^{n} \boldsymbol{w}_{ml}^2 + \sum_{m=1}^{h} \sum_{p=1}^{C} \boldsymbol{v}_{pm}^2 \Big) \tag{12}$$

However, the main issue with using the above quadratic penalty is that all parameters decay exponentially at the same rate and disproportionately penalizes larger weights. Therefore, Weigend et al. [188] proposed the objective shown in Equation 13. When $f(w) := w^2/(1 + w^2)$ this penalty term



(a) Filters are ranked by s_j (b) Prune the smallest filters (c) Prune and retrain

Sorting filters by absolute weights sum for each layer of VGG-16 on CIFAR-10. The x-axis is the filter index divided by the total number of filters. The y-axis is the filter weight sum divided by the max sum value among filters in that layer. (b) Pruning filters with the lowest absolute weights sum and their corresponding test accuracies on CIFAR-10. (c) Prune and retrain for each single layer of VGG-16 on CIFAR-10. Some layers are sensitive and it can be harder to recover accuracy after pruning them.

Figure 3: original source: Li et al. [107]

is small and when large it tends to 1. Therefore, these terms can be considered as approximating the number of non-zero parameters in the network.

$$C(\boldsymbol{w}, \boldsymbol{v}) = \frac{\epsilon}{2}\Big(\sum_{m=1}^{h} \sum_{l=1}^{n} \frac{\boldsymbol{w}_{ml}^2}{1 + \boldsymbol{w}_{ml}^2} + \sum_{m=1}^{h} \sum_{p=1}^{C} \frac{\boldsymbol{v}_{pm}^2}{1 + \boldsymbol{v}_{pm}^2} \Big) \tag{13}$$

The derivative $f'(w) = 2\boldsymbol{w}/(1 + \boldsymbol{w}^2)^2$ computed during backprogation does not penalize large weights as much as Equation 12. However, in the context of recent years where large overparameterized network have shown better generalization when the weights are close to 0, we conjecture that perhaps Equation 13 is more useful in the underparameterized regime. The ϵ controls how the small weights decay faster than large weights. However, the problem of not distinguishing between large weights and very large weights is also an issue. Therefore, Weigend et al. [188] further propose the objective in Equation 14.

$$C(\boldsymbol{w}, \boldsymbol{v}) = \epsilon_1 \sum_{m=1}^{h} \Big(\sum_{l=1}^{n} \frac{\beta \boldsymbol{w}_{ml}^2}{1 + \beta \boldsymbol{w}_{ml}^2} + \sum_{p=1}^{C} \frac{\beta \boldsymbol{v}_{pm}^2}{1 + \beta \boldsymbol{v}_{pm}^2} \Big) + \epsilon_2 \sum_{m=1}^{h} \Big(\sum_{l=1}^{n} \boldsymbol{w}_{ml}^2 + \sum_{p=1}^{C} \boldsymbol{v}_{pm}^2 \Big) \tag{14}$$

Wan et al. [183] have proposed a Gram-Schmidth (GS) based variant of backpropogation whereby GS determines which weights are updated and which ones remain frozen at each epoch.

Li et al. [107] prune filters in CNNs by identifying filters which contribute least to the overall accuracy. For a given layer, sum of the weight magnitudes are computed and since the number of channels is the same across filters, this quantity represents the average of weight value for each kernel. Kernels with weights that have small weight activations will have weak activation and hence these will be pruned. This simple approach leads to less sparse connections and leads to 37% accuracy reduction on average across the models tested while still being close to the original accuracy. Figure 3 shows their figure that demonstrates that pruning filters that have the lowest sum of weight magnitudes correspond to the best maintaining of accuracy.

3.3 Pruning via Loss Sensitivity

Networks can also be pruned by measuring the importance of weights or units by quantifying the change in loss when a weight or unit is removed and prune those which cause the least change in the loss. Many methods from previous decades have been proposed based on this principle [151, 103, 65]. We briefly describe each one below in chronological order.

Skeletonization Mozer and Smolensky [131] estimate which units are least important and deletes them during training. The method is referred to as skeletonization, since it only keeps the units

which preserve the main structure of the network that is required for maintaining good out-of-sample performance. Each weight w in the network is assigned an importance weight α where $alpha = 0$ the weight becomes redundant and $\alpha = 1$ the weight acts as a standard hidden unit.

To obtain the importance weight for a unit, they calculate the loss derivative with respect to α as $\hat{\rho}_i = \partial\mathcal{L}/\alpha_i\big|_{\alpha_i=1}$ where $\mathcal{L}$ in this context is the sum of squared errors. Units are then pruned when $\hat{\rho}_i$ falls below a set threshold. However, they find that $\hat{\rho}_i$ can fluctuate throughout training and so they propose an exponentially-decayed moving average over time to smoothen the volatile gradient and also provide better estimates when the squared error is very small. This moving average is given as,

$$\hat{\rho}_i(t+1) = \beta\hat{\rho}_i(t) + (1-\beta)\frac{\partial\mathcal{L}(t)}{\alpha_i} \tag{15}$$

where $\beta = 0.8$ in their experiments. Applying skeletonization to current DNNs is perhaps be too slow to compute as it was originally introduced in the context of using neural networks with a relatively small amount of parameters. However, assigning importance weights for groups of weights, such as filters in a CNN is feasible and aligns with current literature [190, 5] on structured pruning (discussed in subsection 3.4).

Pruning Weights with *Low Sensitivity* Karnin [90] measure the sensitivity S of the loss function with respect to weights and prune weights with low sensitivity. Instead of removing each weight individually, they approximate S by the sum of changes experienced by the weight during training as

$$S_{ij} = \Big| -\sum_{n=0}^{N-1} \frac{\partial\mathcal{L}}{\partial \boldsymbol{w}_{ij}} \Delta \boldsymbol{w}_{ij}(n) \frac{\boldsymbol{w}_{ij}^f}{\boldsymbol{w}_{ij}^f - \boldsymbol{w}_{ij}^i} \Big| \tag{16}$$

where w^f is the final weight value at each pruning step, w^i is the initial weight after the previous pruning step and N is the number of training epochs. Using backpropagation to compute Δw, $\hat{S}_{ij}$ is expressed as,

$$\hat{S}_{ij} = \Big| -\sum_{n=0}^{N-1} \big[\Delta \boldsymbol{w}_{ij}(n)\big]^2 \frac{\boldsymbol{w}_{ij}^f}{\nabla(\boldsymbol{w}_{ij}^f - \boldsymbol{w}_{ij}^i)} \Big| \tag{17}$$

If the sum of squared errors is less than that of the previous pruning step and if a weight in a hidden layer with the smallest S_{ij} changes less than the previous epoch, then these weights are pruned. This is to ensure that weight with small initial sensitivity are not pruned too early, as they may perform well given more retraining steps. If all incoming weights are removed to a unit, the unit is also removed, thus, removing all outgoing weights from that unit. Lastly, they lower bound the number of weights that can be pruned for each hidden layer, therefore, towards the end of training there may be weights with low sensitivity that remain in the network.

Variance Analysis on Sensitivity-based Pruning Engelbrecht [45] remove weights if its variance in sensitivity is not significantly different from zero. If the variance in parameter sensitivities is not significantly different from zero and the average sensitivity is small, it indicates that the corresponding parameter has little or no effect on the output of the NN over all patterns considered. A hypothesis testing step then uses these variance nullity measures to statistically test if a parameter should be pruned, using the distribution.What needs to be done is to test if the expected value of the sensitivity of a parameter over all patterns is equal to zero. The expectation can be written as $\mathcal{H}_0 : \langle S_{oW,ki}\rangle^2 = 0$ where S_{oW} is the sensitivity matrix of the output vector with respect to the parameter vector $\boldsymbol{W}$ and individual elements $S_{oW,ki}$ refers to the sensitivity of output to perturbations in parameter over all samples. If the hypothesis is accepted, prune the corresponding weight at the (k, i) position, otherwise check $\mathcal{H}_0 : \text{var}(S_{oW,ki}) = 0$ and if this accepted also opt to prune it. They test sum-norm, Euclidean-norm and maximum-norm to compute the output sensitivity matrix. They find that this scheme finds smaller networks than OBD, OBS and standard magnitude-based pruning while maintaining the same accuracy on multi-class classification tasks.

Lauret et al. [101] use a Fourier decomposition of the variance of the model predictions and rank hidden units according to how much that unit accounts for the variance and eliminates based on this

variance-based spectral criterion. For a range of variation $[a_h, b_h]$ of parameter w_h of layer h and N number of training iterations, each weight is varied as $w_h^{(n)} = (b_h + a_h/2) + (b_h - a_h/2)\sin(\omega_h s^(n))$ where $s^{(n)} = 2\pi n/N$ and ω_h is the frequency of w_h and n is the training iteration. The s_h is then obtained by computing the Fourier amplitudes of the fundamental frequency ω_h, the first harmonic up to the third harmonic.

3.3.1 Pruning using Second Order Derivatives

Optimal Brain Damage As mentioned, deleting single weights is computationally inefficient and slow. LeCun et al. [103] instead estimate weight importance by making a local approximation of the loss with a Taylor series and use the 2^{nd} derivative of the loss with respect to the weight as a criterion to perform a type of weight sharing constraint. The objective is expressed as Equation 18

$$\delta\mathcal{L} = \sum_i \cancelto{0}{g_i \delta\breve{w}_i} + \frac{1}{2}\sum_i h_{ii}\delta\breve{w}_i^2 + \frac{1}{2}\sum_{i\neq j} h_{ij}\delta\breve{w}_i\delta w_j + \cancelto{0}{O(\|\breve{W}\|^2)} \tag{18}$$

where $\breve{w}$ are perturbed weights of w, the $\delta\breve{w}_i$'s are the components of $\delta\breve{W}$, g_i are the components of the gradient $\partial\mathcal{L}/\partial\breve{w}_i$ and h_{ij} are the elements of the Hessian $\mathbf{H}$ where $\mathbf{H}_{ij} := \partial^2\mathcal{L}/\partial\breve{w}_i\partial\breve{w}_j$. Since most well-trained networks will have $\mathcal{L} \approx 0$, the 1^{st} term is ≈ 0. Assuming the perturbations on W are small then the last term will also be small and hence LeCun et al. [103] assume the off-diagonal values of $\mathbf{H}$ are 0 and hence $1/2\sum_{i\neq j} h_{ij}\delta\breve{w}_i\delta w_j := 0$. Therefore, $\delta\mathcal{L}$ is expressed as,

$$\delta\mathcal{L} \approx \frac{1}{2}\sum_i h_{ii}\delta\breve{w}_i^2 \tag{19}$$

The 2^{nd} derivatives $\mathbf{h}_{kk}$ are calculated by modifying the backpropogation rule. Since $\boldsymbol{z}_i = f(\boldsymbol{a}_i)$ and $\boldsymbol{a}_i = \sum_j \mathbf{w}_{ij}\boldsymbol{z}_j$, then by substitution $\frac{\partial^2\mathcal{L}}{\partial\mathbf{w}_{ij}^2} = \frac{\partial^2\mathcal{L}}{\partial\mathbf{a}_{ij}^2}z_j$ and they further express the 2^{nd} derivative of the activation output as,

$$\frac{\partial^2\mathcal{L}}{\partial a_i^2} = f'(\boldsymbol{a}_i)^2 - \sum_l \mathbf{w}_{li}^2\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{a}_l^2} - f''(\boldsymbol{a}_i)^2\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{z}_i^2} \tag{20}$$

The derivative of the mean squared error with respect to the to the last linear layer output is then

$$\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{a}_i^2} = 2f'(\boldsymbol{a}_i)^2 - 2(\boldsymbol{y}_i - \boldsymbol{z}_i)f''(\boldsymbol{a}_i) \tag{21}$$

The importance of weight w_i is then $s_k \approx h_{kk}\boldsymbol{w}_k^2/2$ and the portion of weights with lowest s_k are iteratively pruned during retraining.

Optimal Brain Surgeon Hassibi et al. [65] improve over OBD by preserving the off diagonal values of the Hessian, showing empirically that these terms are actually important for pruning and assuming a diagonal Hessian hurts pruning accuracy. To make this Hessian computation feasible, they exploit the recursive relation for calculating the inverse hessian $\mathbf{H}^{-1}$ from training data and the structural information of the network. They denote a weight to be eliminated as $w_q = 0$, $\delta w_q + w_q = 0$ with the objective to minimize the following objective:

$$\min_q \Big\{ \min_{\delta\boldsymbol{w}}\{\frac{1}{2}\delta\boldsymbol{w}^T\cdot\mathbf{H}\cdot\delta\boldsymbol{w}\} \quad s.t \quad \boldsymbol{e}_q^T\cdot\boldsymbol{w} + \boldsymbol{w}_q = 0\Big\} \tag{22}$$

where $\boldsymbol{e}_q$ is the unit vector in parameter space corresponding to parameter w_q. To solve Equation 23 they form a Lagrangian from Equation 22:

$$\mathcal{L} = \frac{1}{2}\delta\boldsymbol{w}^T\cdot\mathbf{H}\cdot\delta\boldsymbol{w} + \lambda(\boldsymbol{e}_q^T\cdot\delta_{\boldsymbol{w}} + \boldsymbol{w}_q) \tag{23}$$

where λ is a Lagrange undetermined multiplier. The functional derivatives are taken and the constraints of Equation 22 are applied. Finally, matrix inversion is used to find the optimal weight change and resulting change in error is expressed as,

$$\delta \boldsymbol{w} = \frac{w_q}{[\mathbf{H}^{-1}_{qq}]} \mathbf{H}^{-1} \boldsymbol{e}_q \quad \text{and} \quad \mathcal{L}_q = \frac{1}{2} \frac{w_q^2}{[\mathbf{H}^{-1}_{qq}]} \tag{24}$$

Defining the first derivative as $\mathbf{X}_k := \frac{f(\boldsymbol{x};\mathbf{W})}{\partial \mathbf{W}}$ the Hessian is expressed as,

$$\mathbf{H} = \frac{1}{P} \sum_{k=1}^{P} \sum_{j=1}^{n} \mathbf{X}_{k,j} \cdot \mathbf{X}_{k,j}^T \tag{25}$$

for an n-dimensional output and P samples. This can be viewed as the sample covariance of the gradient and $\mathbf{H}$ can be recursively computed as,

$$\mathbf{H}^{-1}_{m+1} = \mathbf{H}^{-1}_m + \frac{1}{P} \mathbf{X}^T_{m+1} \cdot \mathbf{X}^T_{m+1} \tag{26}$$

where $\mathbf{H}_0 = \alpha \mathbf{I}$ and $\mathbf{H}_P = \mathbf{H}$. Here $10^{-8} \leq \alpha \geq 10^{-4}$ is necessary to make $\mathbf{H}^{-1}$ less sensitive to the initial conditions. For OBS, $\mathbf{H}^{-1}$ is required and to obtain it they use a matrix inversion formula [88] which leads to the following update:

$$\mathbf{H}^{-1}_{m+1} = \mathbf{H}^{-1}_m - \frac{\mathbf{H}^{-1}_m \cdot \mathbf{X}_{m+1} \cdot \mathbf{X}^T_{m+1} \cdot \mathbf{H}^{-1}_m}{P + \mathbf{X}^{-1}_{m+1} \cdot \mathbf{H}^{-1}_m \cdot \mathbf{X}_{m+1}} \quad \text{where} \quad \mathbf{H}_0 = \alpha \mathbf{I}, \quad \mathbf{H}_P = \mathbf{H} \tag{27}$$

This recursion step is then used as apart of Equation 24, can be computed in one pass of the training data $1 \leq m \leq P$ and computational complexity of $\mathbf{H}$ remains the same as $\mathbf{H}^{-1}$ as $\mathcal{O}(Pn^2)$. Hassibi et al. [65] have also extended their work on approximating the inverse hessian [64] to show that this approximation works for any twice differentiable objective (not only constrained to sum of squared errors) using the Fisher's score.

Other methods to Hessian approximation include dividing the network into subsets to use block diagonal approximations and eigen decomposition of $\mathbf{H}^{-1}$ [65] and principal components of $\mathbf{H}^{-1}$ [105] (unlike aforementioned approximations, Levin et al. [105] do not require the network to be trained to a local minimum). However the main drawback is that the Hessian is relatively expensive to compute for these methods, including OBD. For n weights, the Hessian requires $\mathcal{O}(n^2/2)$ elements to store and performs $\mathcal{O}(Pn^2)$ calculations per pruning step, where P is total number of pruning steps.

Taylor Expansion Criterion for Pruning Molchanov et al. [130] also use a Taylor expansion (TE) as a criterion to prune by choosing a subset of weights W_s which have a minimal change on the cost function. They also add a regularization term that explicitly regularize the computational complexity of the network. Equation 28 shows how the absolute cost difference between the original network cost with weights w and the pruned network with w' weights is minimized such that the number of parameters are decreased where $|| \cdot ||_0$ denotes the 0-norm bounds the number of non-zero parameters W_s.

$$\min_{\mathbf{W}'} |\mathcal{C}(D|\mathbf{W}') - \mathcal{C}(D|\mathbf{W})| \quad \text{s.t.} \quad |\mathbf{W}'|_0 \leq \mathbf{W}_s \tag{28}$$

Unlike OBD, they keep the absolute change $|y|$ resulting from pruning, as the variance σ_y^2 is non-zero and correlated with stability of the $\partial C / \partial h$ throughout training, where $\boldsymbol{h}$ is the activation of the hidden layer. Under the assumption that samples are independent and identically distributed, $\mathbb{E}(|y|) = \sigma\sqrt{2}/\sqrt{\pi}$ where σ is the standard deviation of y, known as the expected value of the half-normal distribution. So, while y tends to zero, the expectation of $|y|$ is proportional to the variance of y, a value which is empirically more informative as a pruning criterion.

They rank the order of filters pruned using the TE criterion and compare to an oracle rank (i.e the best ranking for removing pruned filters) and find that it has higher spearman correlation to the oracle when compared against other ranking schemes. This can also be used to choose which filters should be transferred to a target task model. They compute the importance of neurons or filters z by estimating the mutual information with target variable $\text{MI}(z; y)$ using information gain $IG(y|z) = \mathcal{H}(z) + \mathcal{H}(y) - \mathcal{H}(z, y)$ where $\mathcal{H}(z)$ is the entropy of the variable z, which is quantized to make this estimation tractable.

Fisher Pruning Theis et al. [175] extend the work of Molchanov et al. [130] by providing a motivation for the pruning scheme and provide computational cost estimates during pruning as adjacent layers are successively being pruned. Unlike OBD and OBS, they use, fisher pruning as it is more efficient since the gradient information is already computed during the backward pass.

As before, the 2^{nd} TE approximates the loss with respect to w is made and the fisher information computed during backpropogation is used as the criterion. Unlike Molchanov et al. [130],

The gradient can be formulated as Equation 29, where $\mathcal{L}(w) = \mathbb{E}_P[-\log Q_{\boldsymbol{w}}(y|\boldsymbol{x})]$, d represents a change in parameters, P is the underlying distribution, $Q_w(y|x)$ is the posterior from the model H is the Hessian matrix.

$$g = \nabla\mathcal{L}(w), \quad \mathbf{H} = \nabla^2\mathcal{L}(w), \quad \mathcal{L}(w+d) - \mathcal{L}(w) \approx \boldsymbol{g}^T d + \frac{1}{2}\boldsymbol{d}^T \mathbf{H}\boldsymbol{d} \tag{29}$$

$$\mathcal{L}(\mathbf{W} - \mathbf{W}_k \boldsymbol{e}_i) - \mathcal{L}(\mathbf{W}) + \beta \cdot (C(W - \mathbf{W}_k \boldsymbol{e}_i) - C(\mathbf{W})) \tag{30}$$

3.4 Structured Pruning

Since standard pruning leads to non-structured connectivity, structured pruning can be used to reduce speed and memory since hardware is more amenable to dealing with dense matrix multiplications, with little to no non-zero entries in matrices and tensors. CNNs in particular are suitable for this type of pruning since they are made up of sparse connections. Hence, below we describe some work that use group-wise regularizers, structured variational, Adversarial Bayesian methods to achieve structured pruning in CNNs.

3.4.1 Structured Pruning via Weight Regularization

Group Sparsity Regularization Group sparse regularizers enforce a subset of weight groupings, such as filters in CNNs, to be close to zero when trained using stochastic gradient descent. Consider a convolutional kernel represented as a tensor $K(i, j, s, :)$, the group-wise $\ell_2, 1$-norm is given as

$$\omega_{2,1}(K) = \lambda \sum_{i,j,s} ||\Gamma_{ijs}|| = \lambda \sum_{ijs} \sqrt{\sum_{t=1}^{T} K(i, j, s, t)^2} \tag{31}$$

where Γ_{ijs} is the group of kernel tensor entries $K(i, j, s, :)$ where (i, j) are the pixel of i-th row and j-th column of the image for the s-th input feature map. This regularization term forces some Γ_{ijs} groups to be close to zero, which can be removed during retraining depending on the amount of compression that the practitioner predefines.

Structured Sparsity Learning Wen et al. [190] show that their proposed structural regularization can reduce a ResNet architecture with 20 layers to 18 with 1.35 percentage point accuracy increase on CIFAR-10, which is even higher than the larger 32 layer ResNet architecture. They use a group lasso regularization to remove whole filters, across channels, shape and depth as shown in Figure 4.

Equation 32 shows the loss to be optimized to remove unimportant filters and channels, where $\mathbf{W}(l)_{n_l, c_l, :, :}$ is the c-th channel of the l-th filter for a collection of all weights $\mathbf{W}$ and $||\cdot||$ is the group Lasso regularization term where $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(\boldsymbol{w}^{(g)}\right)^2}$ and $|\boldsymbol{w}^{(g)}|$ is the number of weights in $\boldsymbol{w}^{(g)}$.

Since zeroing out the l-th filter leads to the feature map output being redundant, it results in the $l+1$ channel being zeroed as well. Hence, structured sparsity learning is carried out for both filters and channels simultaneously.

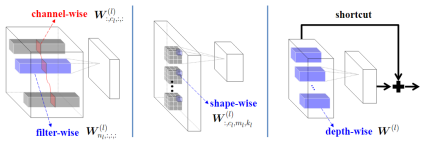


Figure 4: original source: Wen et al. [190]: Structured Sparsity Learning

$$\mathcal{L}(\mathbf{W}) = \mathcal{L}_D(\mathbf{W}) + \lambda_n \cdot \sum_{l=1}^{L} \Big(\sum_{n_l=1}^{N} ||\mathbf{W}^{(l)}_{m_l,:,:,:}||_g \Big) + \lambda_c \cdot \sum_{l=1}^{L} \Big(\sum_{c_l=1}^{C_l} ||\mathbf{W}^{(l)}_{c_l,:,:,:}||_g \Big) \tag{32}$$

3.4.2 Structured Pruning via Loss Sensitivity

Structured Brain Damage The aforementioned OBD has also been extended to remove groups of weights using group-wise sparse regularizers (GWSR) Lebedev and Lempitsky [102]. In the case of filters in CNNs, this results in smaller and reshaped matrices, leading to smaller and faster CNNs. The GWSR is added as a regularization term during retraining a pretrained CNN and after a set number of epochs, the groups with smallest ℓ_2 norm are deleted and the number of groups are predefined as $\tau \in [0, 1]$ (a percentage of the size of the network). However, they find that when choosing a value for τ, it is difficult to set the regularization influence term λ and can be time consuming manually tuning it. Moreover when τ is small, the regularization strength of λ is found to be too heavy, leading to many weight groups being biased towards 0 but not being very close to it. This results in poor performance as it becomes more unclear what groups should be removed. However, the drop in accuracy due to this can be remedied by further retraining after performing OBD. Hence, retraining occurs on the sparse network without using the GWSR.

3.4.3 Sparse Bayesian Priors

Sparse Variational Dropout Seminal work, such as the aforementioned Skeletonization [131] technique has essentially tried to learn weight saliency. Variational dropout (VD), or more specifically Sparse Variational Dropout [(SpVD) 129], learn individual dropout rates for each parameter in the network using varitaional inference (VI). In Sparse VI, sparse regularization is used to force activations with high dropout rates (unlike the original VD [95] where dropout rates are bound at 0.5) to go to 1 leading to their removal. Much like other sparse Bayes learning algorithms, VD exhibits the Automatic relevance determination (ARD) effect[9]. Molchanov et al. [129] propose a new approximation to the KL-divergence term in the VD objective and also introduce a way to reduce variance in the gradient estimator which leads to faster convergence. VI is performed by minimizing the bound between the variational Gaussian prior $q_\phi(w)$ and prior over the weight $p(w)$ as,

$$\mathcal{L}(\phi) = \max_{\phi} \mathcal{L}_D - D_{\text{KL}}\Big(q_\phi(w)||p(w)\Big) \quad \text{where} \quad \mathcal{L}_D(\phi) = \sum_{n=1}^{N} \mathbb{E}_{q_\phi(w)}\Big[\log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}_n)\Big] \tag{33}$$

They use the reparameterization trick to reduce variance in the gradient estimator when $\alpha > 0.5$ by replacing multiplicative noise $1 + \sqrt{\alpha_{ij}} \cdot \epsilon_{ij}$ with additive noise $\sigma_{ij} \cdot \epsilon_{ij}$, where $\epsilon_{ij} \sim \mathcal{N}(0; 1)$ and $\sigma^2_{ij} = \alpha_{ij} \cdot \theta^2_{ij}$ is tuned by optimizing the variational lower bound w.r.t θ and σ. This difference with the original VD allow weights with high dropout rates to be removed.

Since the prior and approximate posterior are fully factorized, the full KL-divergence term in the lower bound is decomposed into a sum:

$$D_{\text{KL}}(q(\mathbf{W}|\theta, \alpha)||p(\mathbf{W})) = \sum_{ij} D_{\text{KL}}(q(w_{ij}|\theta_{ij}, \alpha_{ij})||p(w_{ij})) \tag{34}$$

[9] Automatic relevance determination provides a data-dependent prior distribution to prune away redundant features in the overparameterized regime i.e more features than samples

Since the uniform log-prior is an improper prior, the KL divergence is only computed up to an additional constant [95].

$$-D_{\mathrm{KL}}(q(w_{ij}|\theta_{ij},\alpha_{ij})||p(w_{ij})) = \frac{1}{2}\log\alpha_{ij} - E \sim N(1,\alpha_{ij})\log|\cdot| + C \tag{35}$$

In the VD model this term is intractable, as the expectation $E \sim N(1,\alpha_{ij})\log|\cdot|_i n$ cannot be computed analytically [95]. Hence, they approximate the negative KL. The negative KL increases as α_{ij} increases which means the regularization term prefers large values of α_{ij} and so the correspond weight w_{ij} is dropped from the model. Since using SVD at the start of training tends to drop too many weights early since the weights are randomly initialized, SVD is used after an initial pretraining stage and hence this is why we consider it a pruning technique.

Bayesian Structured Pruning Structured pruning has also been achieved from a Bayesian view [120] of learning dropout rates. Sparsity inducing hierarchical priors are placed over the units of a DNN and those units with high dropout rates are pruned. Pruning by unit is more efficient from a hardware perspective than pruning weights as the latter requires priors for each individual weight, being far more computationally expensive and has the benefit of being more efficient from a hardware perspective as whole groups of weights are removed.

If we consider a DNN as $p(D|w) = \prod_{i=1}^{N} p(y_i|x_i,w)$ where x_i is a given input sample with a corresponding target y_i, w are the weights of the network, governed by a prior distribution $p(w)$. Since computing the posterior $p(w|D) = p(D|w)p(w)/p(D)$ explicitly is intractactble, $p(w)$ is approximated with a simpler distribution, such as a Gaussian $q(w)$, parameterized by variational parameters ϕ. The variational parameters are then optimized as,

$$\mathcal{L}_E = \mathbb{E}_{q_\phi(w)}[\log p(D|w)], \quad \mathcal{L}_C = \mathbb{E}_{q_\phi(w)}[\log p(w)] + \mathcal{H}(q_\phi(w)) \tag{36}$$

$$L(\phi) = \mathcal{L}_E + \mathcal{L}_C \tag{37}$$

where $\mathcal{H}(\cdot)$ denotes the entropy and $\mathcal{L}(\phi)$ is known as the evidence-lower-bound (ELBO). They note that $\mathcal{L}_E$ is intractable for noisy weights and in practice Monte Carlo integration is used. When the simpler $q_\phi(w)$ is continuous the reparameterization trick is used to backpropograte through the deterministic part ϕ and Gaussian noise $\epsilon \sim N(0,\sigma^2 I)$. By substituting this into Equation 36 and using the local reparameterization trick [95] they can express $\mathcal{L}(\phi)$ as

$$\mathcal{L}(\phi) = \mathbb{E}_p(\epsilon)[\log p(D|f(\phi,\epsilon))] + \mathbb{E}_{q_{\epsilon(w)}}[\log p(w)] + \mathcal{H}(q_{\phi(w)}), \quad \text{s.t} \quad w = f(\phi,\epsilon) \tag{38}$$

with unbiased stochastic gradient estimates of the ELBO w.r.t the variational parameters ϕ. They use mixture of a log-uniform prior and a half-Cauchy prior for $p(w)$ which equates to a horseshoe distribution [20]. By minimizing the negative KL divergence between the normal-Jeffreys scale prior $p(z)$ and the Gaussian variational posterior $q_\phi(z)$ they can learn the dropout rate $\alpha_i = \sigma^2 z_i/\mu_2 z_i$ as

$$-D_{\mathrm{KL}}(\phi(z)||p(z)) \approx A\sum_i (k_1\sigma(k_2 + k_3\log\alpha_i) - 0.5m(-\log\alpha_i) - k_1) \tag{39}$$

where $\sigma(\cdot)$ is the sigmoid function, $m(\cdot)$ is the softplus function and $k_1 = 0.64$, $k_2 = 1.87$ and $k_3 = 1.49$. A unit i is pruned if its variational dropout rate does not exceed threshold t, as $\log\alpha_i = (\log\sigma^2 zi - \log\mu_2 z_i) \geq t$.

It should be mentioned that this prior parametrization readily allows for a more flexible marginal posterior over the weights as we now have a compound distribution,

$$q_\phi(W) = \int q_\phi(W|z)q_\phi(z)dz \tag{40}$$

Pruning via Variational Information Bottleneck Dai et al. [34] minimize the variational lower bound (VLB) to reduce the redundancy between adjacent layers by penalizing their mutual information to ensure each layer contains useful and distinct information. A subset of neurons are kept while the remaining neurons are forced toward 0 using sparse regularization that occurs as apart of their variational information bottleneck (VIB) framework. They show that the sparsity inducing regularization has advantages over previous sparsity regularization approaches for network pruning.

Equation 41 shows the objective for compressing neurons (or filters in CNNs) where γ_i controls the amount of compression for the i-th layer and L is a weight on the data term that is used to ensure that for deeper networks the sum of KL factors does not result in the log likelihood term outweighed when finding the globally optimal solution.

$$\mathcal{L} = \sum_{i=1}^{L} \gamma_i \sum_{j=1}^{r_i} \log\left(1 + \frac{\mu_{i,j}^2}{\sigma_{i,j}^2}\right) - L\mathbb{E}_{\{\boldsymbol{x},y\}\sim D,\ \boldsymbol{h}\sim p(\boldsymbol{h}|\boldsymbol{x})}\Big[\log q(\boldsymbol{y}|\boldsymbol{h}_L)\Big] \tag{41}$$

L naturally arises from the VIB formulation unlike probabilistic networks models. The $\log(1+u)$ in the KL term is concave and non-decreasing for range $[0, \infty]$ and therefore favors solutions that are sparse with a subset of parameters exactly zero instead of many shrunken ratios $\alpha_{i,j} : \mu_{i,j}^2 \sigma_{i,j}^{-2},\ \forall i, j$.

Each layer is sampled $\epsilon_i \sim \mathcal{N}(0, I)$ in the forward pass and $\boldsymbol{h}_i$ is computed. Then the gradients are updated after backpropogation for $\{\mu_i, \sigma_i \mathbf{W}_i\}_{i=1}^{L}$ and output weights $\mathbf{W}_y$.

Figure 5 shows the conditional distribution $p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})$ and $\boldsymbol{h}_i$ sampled by multiplying $f_i(\boldsymbol{h}_{i-1})$ with a random variable $\boldsymbol{z}_i := \mu_i + \epsilon_i \circ \sigma_i$.

They show that when using VIB network, the mutual information increases between $\boldsymbol{x}$ and $\boldsymbol{h}_1$ as it initially begins to learn and later in training the mutual information begins to drop as the model enters the compression phase. In constrast, the mututal information for the original stayed consistently high tending towards 1.

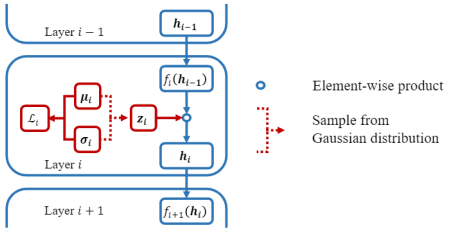


Figure 5: original source Dai et al. [34]: Variational Information Structure

Generative Adversarial-based Structured Pruning Lin et al. [113] extend beyond pruning well-defined structures, such as filters, to more general structures which may not be predefined in the network architecture. They do so applying a soft mask to the output of each structure in a network to be pruned and minimize the mean squared error with a baseline network and also a minimax objective between the outputs of the baseline and pruned network where a discriminator network tries to distinguish between both outputs. During retraining, soft mask weights are learned over each structure (i.e filters, channels,) with a sparse regularization term (namely, a fast iterative shrinkage-thresholding algorithm) to force a subset of the weights of each structure to go to 0. Those structures which have corresponding soft mask weight lower than a predefined threshold are then removed throughout the adversarial learning. This soft masking scheme is motivated by previous work [112] that instead used hard thresholding using binary masks, which results in harder optimization due to non-smoothness. Although they claim that this sparse masking can be performed with label-free data and transfer to other domains with no supervision, the method is largely dependent on the baseline (i.e teacher network) which implicitly provides labels as it is trained with supervision, and thus it pruned network transferability is largely dependent on this.

3.5 Search-based Pruning

Search-based techniques can be used to search the combinatorial subset of weights to preserve in DNNs. Here we include pruning techniques that don't rely on gradient-based learning but also evolutionary algorithms and SMC methods.

3.5.1 Evolutionary-Based Pruning

Pruning using Genetic Algorithms The basic procedure for Genetic Algorithms (GAs) in the context of DNNs is as follows; (1) generate populations of parameters (or *chromosones* which are binary strings), (2) keep the top-k parameters that perform the best (referred to as tournament selection) according to a predefined *fitness* function (e.g classification accuracy), (3) randomly mix (i.e cross over) between the parameters of different sets within the top-k and perturb a portion of the resulting parameters (i.e mutation) and (4) repeat this procedure until convergence. This procedure can be used to find a subset of the DNN network that performs well.

Whitley et al. [191] use a GA to find the optimal set of weights which involves connecting and reconnecting weights to find mutations that lead to the highest fitness (i.e lowest loss). They define the number of backpropogation steps as $ND + B$ where B is the baseline number of steps, N is the number of weights pruned and D is the increase in number of backpropgation steps. Hence, if the network is heavily pruned the network is allocated more retraining steps. Unlike standard pruning techniques, weights can be reintroduced if they are apart of combination that leads to a relatively good fitness score. They assign higher reward to network which more heavily pruned, otherwise referred to as *selective pressure* in the context of genetic algorithms.

Since the cross-over operation is not specific to the task by default, interference can occur among related parameters in the population which makes it difficult to find a near optimal solution, unless the population is very large (i.e exponential with respect to the number of features). Cantu-Paz [19] identify the relationship between variables by computing the joint distribution of individuals left after tournament selection and use this sub-population to generate new members of the population for the next iteration. This is achieved using 3 distribution estimation algorithms (DEA). They find that DEAs can improve GA-based pruning and that in pruned networks using GA-based pruning results in faster inference with little to no difference in performance compared to the original network.

Recently, Hu et al. [80] have pruned channels from a pretrained CNN using GAs and performed knowledge distillation on the pruned network. A kernel is converted to a binary string K with a length equal to the number of channels for that kernel. Then each channel is encoded as 0 or 1 where channels with a 0 are pruned and the n-th kernel K_n is represented a a binary series after sampling each bit from a Bernoulli distribution for all C channels. Each member (i.e channels) in the population is evaluated and top-k are kept for the next generation (i.e iteration) based on the fitness score where k corresponds to the total amount of pruning. The Roulette Wheel algorithm is used as the selection strategy [52] whereby the n-th member of the m-th generation $I_{m,n}$ has a probability of selection proportional to its fitness relative to all other members. This can simply be implemented by inputting all fitness scores for all members into a softmax. To avoid members with high fitness scores losing information post mutation and cross-over, they also copy the highest fitness scoring members to the next generation along with their mutated versions.

The main contribution is a 2-stage fitness scoring process. First, a local TS approximation of a layer-wise error function using the aforementioned OBS objective [42] (recall that OBS mainly revolves around efficient Hessian approximation) is used sequentially from the first layer to the last, followed by a few epochs of retraining to restore the accuracy of the pruned network. Second, the pruned network is distilled usin a cross-entropy loss and regularization term that forces the features maps of the pruned network to be similar to the distilled model, using an attention map to ensure both corresponding layer feature maps are of the same and fixed size. They achieve SoTA on ImageNet and CIFAR-10 for VGG-16 and ResNet CNN architectures using this approach.

Pruning via Simulated Annlealing Noy et al. [138] propose to reduce search time for searching neural architectures by relaxing the discrete search to continuous that allows for a differentiable simulated annealing that is optimized using gradient descent (following from the DARTS [115] approach). This leads to much faster solutions compared to using black-box search since optimizing over the continuous search space is an easier combinatorial optimization problem that in turn leads to faster convergence. This pruning technique is not strictly consider compression in its standard definition, as it prunes during the initial training period as opposed to pruning after pretraining. This falls under the category of neural architecture search (NAS) and here they use an annealing schedule that controls the amount of pruning during NAS to incrementally make it easier to search for sub-modules that are found to have good performance in the search process. Their (0, δ)-PAC

theorem guarantees under few assumptions (see paper for further details on these assumptions) that this anneal and prune approach prunes less important weights with high probability.

3.5.2 Sequential Monte Carlo & Reinforcement Learning Based Pruning

Particle Filter Based Pruning Anwar et al. [5] identifies important weights and paths using particle filters where the importance weight of each particle is assigned based on the misclassification rate with corresponding connectivity pattern. Particle filtering (PF) applies sequential Monte Carlo estimation with particle representing the probability density where the posterior is estimated with a random sample and parameters that are used for posterior estimation. PF propogates parameters with large magnitudes and deletes parameters with the smallest weight in re-sampling process, similar to MBP. They use PF to prune the network and retrain to compensate for the loss in performance due to PF pruning. When applied to CNNs, they reduce the size of kernel and feature map tensors while maintaining test accuracy.

Particle Swarm Optimized Pruning Particle Swarm Optimization (PSO) has also been combined with correlation merging algorithm (CMA) for pruning [177]. Equation 42 shows the PSO update formula where the velocity $\mathbf{V}_{id}$ for i-th position of particle $\mathbf{X}_i d$ (i.e a parameter vector in a DNN) at the d-th iteration,

$$\mathbf{V}_{id} := \mathbf{V}_{id} + c_1 u(\boldsymbol{P}_{id} - \boldsymbol{X}_{id}) + c_2 u(\boldsymbol{P}_{gd} - \boldsymbol{X}_{id}), \quad \text{where} \quad \boldsymbol{X}_{id} = \boldsymbol{X}_{id} + \mathbf{V}_{id} \tag{42}$$

where $u \sim \text{Uniform}(0, 1)$ and c_1, c_2 are both learning rates, corresponding to the influence social and cognition components of the swarm respectively [91]. Once the velocity vectors are updated for the DNN, the standard deviation is computed for the i-th activation as $s_i = \sum_{p=1}^{n} (\mathbf{V}_{ip} - \bar{\mathbf{V}}_i)^2$ where $\bar{v}_i$ is the mean value of $\mathbf{V}_i$ over training samples.

Then compute Pearson correlation coefficient between the i-th an j-th unit in the hidden layer as $\mathbf{C}_{ij} = (\mathbf{V}_{ip}\mathbf{V}_{jp} - n\bar{\mathbf{V}}_i\bar{\mathbf{V}}_j)/\boldsymbol{S}_i\boldsymbol{S}_j$ and if $\mathbf{C}_{ij} > \tau_1$ where τ is a predefined threshold, then merge both units, delete the j-th unit and update the weights as,

$$\mathbf{W}_{ki} = \mathbf{W}_{ki} + \alpha\mathbf{W}_{ki} \quad \text{and} \quad \mathbf{W}_{kb} = \mathbf{W}_{kb} + \beta\mathbf{W}_k \tag{43}$$

where,

$$\alpha = \frac{\mathbf{V}_{ip}\mathbf{V}_{jp} - n\bar{\mathbf{V}}_i\bar{\mathbf{V}}_j}{\sum_{n=1}^{p} \mathbf{V}_{ip}\mathbf{V}_{jp} - \bar{\mathbf{V}}_i^2}, \quad \beta = \bar{\mathbf{V}}_j - \alpha\bar{\mathbf{V}}_i \tag{44}$$

and $\mathbf{W}_{ki}$ connects the last hidden layer to output unit k. If the standard deviation of unit i is less than τ_2 then it is combined with the output unit k. Finally, remove unit j and update the bias of the output unit k as $\mathbf{W}_{kb} = \mathbf{W}_{kb} + \bar{\boldsymbol{V}}_i\mathbf{W}_{ki}$. This process is repeated until a maximally compressed network than maintains performance similar to the original network is found.

Automated Pruning AutoML [68] use RL to improve the efficiency of model compression performance by exploiting the fact that the sparsity of each layer is a strong signal for the overall performance. They search for a compressed architecture in a continuous space instead of searching over a discrete space. A continuous compression ratio control strategy is employed using an actor critic model (Deep Deterministic Policy Gradient [168]) which is known to be relatively stable during training, compared to alternative RL models, due lower variance in the gradient estimator. The DDPG processes each consecutive layer, where for the t-th layer L_t, the network receives a layer embedding t that encodes information of this layer and outputs a compression ratio a_t and repeats this process from the first to last layer. The resulting pruned network is evaluated without fine-tuning, avoiding retraining to improve computational cost and time. During training, they fine-tune best explored model given by the policy search. The MBP ratio is constrained such that the compressed model produced by the agent is below a resource constrained threshold in resource constrained case. Moreover, the maximum amount of pruning for each layer is constrained to be less than 80%, When the focus is to instead maintain accuracy, they define the reward function to incorporate accuracy and the available hardware resources.

By only requiring 1/4 number of the FLOPS they still manage to achieve a 2.7% increase in accuracy for MobileNet-V1. This also corresponds to a 1.53 times speed up on a Titan Xp GPU and 1.95 times speed up on Google Pixel 1 Android phone.

4 Low Rank Matrix & Tensor Decompositions

DNNs can also be compressed by decomposing the weight tensors (2^{nd} order tensor in the case of a matrix) into a lower rank approximation which can also removed redundancies in the parameters. Many works on applying TD to DNNs have been predicated on using SVD [194, 156, 195, 195, 136]. Hence, before discussing different TD approaches, we provide an introduction to SVD.

A matrix $\mathbb{A} \in \mathbb{R}^{m \times n}$ of full rank r can be decomposed as $\mathbf{A} = \mathbf{WH}$ where $\mathbf{W} \in \mathbb{R}^{m \times r}$ and $\mathbf{H} \in \mathbb{R}^{r \times n}$. The change in space complexity as $\mathcal{O}(\text{mn}) \rightarrow \mathcal{O}(r(m+n))$ at the expense of some approximation error after optimizing the following objective,

$$\min_{\mathbf{W},\mathbf{H}} \frac{1}{2} ||\mathbf{A} - \mathbf{WH}||_F^2 \tag{45}$$

where for a low rank $k < r$, $\mathbf{W} \in \mathbb{R}^{m \times k}$ and $\mathbf{H} \in \mathbb{R}^{k \times n}$ and $|| \cdot ||_F$ is the Frobenius norm.

A common technique for achieving this low rank TD is Singular Value Decomposition (SVD). For orthogonal matrices $\mathbf{U} \in \mathbb{R}^{m \times r}$, $\mathbf{V} \in \mathbb{R}^{n \times r}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{r \times r}$ of singular values, we can express $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{46}$$

where if $k < r$ then this is called truncated SVD. The nonzero elements of Σ are the sorted in decreasing order and the top k $\Sigma_k \in \mathbb{R}^{k \times k}$ are used as $\mathbf{A} \approx \mathbf{U}_k \Sigma_k \mathbf{V}_k^T$.

Randomized SVD [60] has also been introduced for faster approximation using ideas from random matrix theory. An approximation of the range $\mathbf{A}$ by finding $\mathbf{Q}$ with r othornomal columns and $\mathbf{A} \approx \mathbf{QQ}^T\mathbf{A}$. Then the SVD is found by constructing a matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$ and SVD is instead computed on $\mathbf{B}$ as before using Equation 46. Since $\mathbf{A} \approx \mathbf{QB}$, we can see $\mathbf{U} = \mathbf{QS}$ computes a LRD $\mathbf{A} \approx U\mathbf{SV}^T$

Then as $\mathbf{A} \approx \mathbf{QQ}^T\mathbf{A} = \mathbf{Q}(\mathbf{S}\Sigma\mathbf{V}^T)$, we see that taking $\mathbf{U} = \mathbf{QS}$, we have computed a low rank approximation $\mathbf{A} \approx U\mathbf{SV}^T$. Approximating $\mathbf{Q}$ is achieved by forming a Gaussian random matrix $\omega \in \mathbb{R}^{n \times l}$ and computing $\mathbf{Z} = \mathbf{A}\omega$, and using QR decomposition of $\mathbf{Z}, \mathbf{QR} = \mathbf{Z}$, then $\mathbf{Q} \in \mathbb{R}^{m \times l}$ has columns that are an orthonormal basis for the range of $\mathbf{Z}$.

Numerical precision is maintained by taking intermediate QR and LU decompositions during o power iterations of $\mathbf{AA}^T$ to reduce Y's spectrum because if the singular values of $\mathbf{A}$ are Σ, then the singular values of $(\mathbf{AA}^T)^o$ are Σ^{2o+1}. With each power iteration the spectrum decays exponentially, therefore it only requires very few iterations.

4.1 Tensor Decomposition

Generalizing $\mathbf{A}$ to higher order tensors, which we can refer to as an ℓ-way array $\mathcal{A} \in \mathbb{R}^{n_a \times n_b \dots \times n_x}$, the aim is to find the components $\mathcal{A} = \sum_i^r a \circ b \circ x = [[\mathbf{A}, \mathbf{B} \dots, \mathbf{X}]]$.

Before discussing the TD we first define three important types of matrix products used in tensor computation:

- The Kronecker product between two arbitrarily-sized matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$,$\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{(I) \times (JL)}$, is a generalization of the outer product from vectors to 3 matrices $\mathbf{A} \otimes \mathbf{B} := [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \boldsymbol{a}_2 \otimes \boldsymbol{b}_2, \dots \boldsymbol{a}_J \otimes \boldsymbol{b}_{L-1}, \boldsymbol{a}_J \otimes \boldsymbol{b}_L]$.
- The Khatri-Rao product between two matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$, $\mathbf{AB} \in \mathbb{R}^{(IJ) \times K}$, corresponds to the column-wise Kronecker product. $\mathbf{AB} := [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \boldsymbol{a}_2 \otimes b_2 \dots \boldsymbol{a}_K \otimes \boldsymbol{b}_K]$.
- The Hadamard product is the elementwise product between 2 matrices **A, B** $\in \mathbb{R}^{I \times J}$ and $\mathbf{A} * \mathbf{B} \in \mathbb{R}^{I \times J}$.

These products are used when performing Canonical Polyadic [(CP) 74], Tucker decompositions [178], Tensor Train [TT 139] to find the factor matrices $\mathcal{X} := [[\mathbf{A}, \mathbf{B} \ldots, \mathbf{C}]]$. For the sake of simplicity we'll proceed with 3-way tensors. As before in Equation 45, we can express the optimization objective as

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i,j,k} ||\boldsymbol{x}_{ijk} - \sum_{l} \boldsymbol{a}_{il}\boldsymbol{b}_{jl}\boldsymbol{c}_{\text{kl}}||^2 \tag{47}$$

Since the components $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are not orthogonal, we cannot compute SVD as was the case for matrices. The rank r of $\mathbf{A}$ is also NP-hard and the solutions found for lower rank approximations may not be apart of the solution for higher ranks. Unlike, when you rotate the row or column vectors of a matrix and apply dimensionality reduction (e.g PCA) and still get the same solution, this is not the case for TD. Unlike matrices where there can be many low rank matrices, a tensor is requires to have a low-rank matrix that is compatible for all tensor slices. This interconnection between different slices results in tensor being more restrictive and hence the for weaker uniqueness conditions.

One way to perform TD using Equation 47 is to use alternating least squares (ALS) which involves minimizing $\min_{\mathbf{A}}$ while fixing $\mathbf{B}, \mathbf{C}$ and repeating this for $\min_{\mathbf{B}}$ and $\min_{\mathbf{C}}$. ALS is suitable because it is a nonconvex optimization problem but with convex subproblems. CP can be generalized to different objectives apart from the squared loss, such as Rayleigh (when entries are non-negative), Boolean (entries are binary) and Poisson (when entries are counts) losses. Similar to randomized SVD described in the previous subsection, they have also been successful when scaling up TD.

With this introduction, we now move onto how low rank TD has been applied to DNNs for reducing the size of large weight matrices.

4.2 Applications of Tensor Decomposition to Self-Attention and Recurrent Layers

4.2.1 Block-Term Tensor Decomposition (BTD)

Block-Term Tensor Decomposition [(BTD) 37] combines CP decomposition and Tucker decomposition. Consider an $n-th$ order tensor as $\mathcal{A} \in \mathbb{R}^{A_1 x \ldots x A_n}$ that can be decomposed into N block terms and each block consist of k elements between a core tensor $\mathcal{G}_n \in \mathbb{R}^{G_1 \times \ldots G_d}$ and d and factor matrices $\mathcal{C}_n^{(k)} \in \mathbb{R}^{A_k \times G_k}$ along the k-th dimension where $n \in [1, N]$ and $k \in [1, d]$ [37]. BTD can then be defined as,

$$\mathcal{A} = N \sum_{n=1} \mathcal{G}_n \otimes \mathcal{C}_n^{(1)} \otimes \mathcal{C}_n^{(2)} \ldots \mathcal{C}_n^{(d)} \tag{48}$$

The N here is the CP-rank, G_1, G_2, G_3 is the Tucker-rank and d the core-order.

BTD RNNs Ye et al. [198] also used BTD to learn small and dense RNNs by first tensorizing the RNN weights and inputs to a 3-way tensor as $\mathcal{X}$ and $\mathcal{W}$ respectively. BTD is then performed on the weights $\mathcal{W}$ and a tensorized backpropogation is computed when updating the weights. The core-order d is important when deciding the total number of parameters and they recommend $d = [3, 5]$ which is a region that corresponds to orders of magnitude reduction in the number of parameters. When $d > 5$ the number of parameters begins to increase again since the number of parameters is defined as $p_{BTD} = N \sum_{k=1}^{d} Y_k Z_k R + R^d$ where Y_k is the row of the k-th matrix, Z_k the number of columns and $\mathbb{R}^d$ is responsible for exponential increase in the core tensors. Moreover, too high d will result in the loss of spatial information. For a standard forward and backward pass, the time complexity and memory requires is $\mathcal{O}(YZ)$ for W. For BTD-RNN, the time complexity is $\mathcal{O}(NdYR^dZ_{\text{max}})$ on the forward pass and $\mathcal{N}\lceil^{\in}\mathcal{Y}\mathcal{R}\rceil\mathcal{Z}_{\text{max}}$ on the backward pass where J. For spatial complexity it is $\mathcal{O}(R^dY)$ on both passes. They find significant improvements over an LSTM baseline network and improvements over a Tensor-Train LSTM [199].

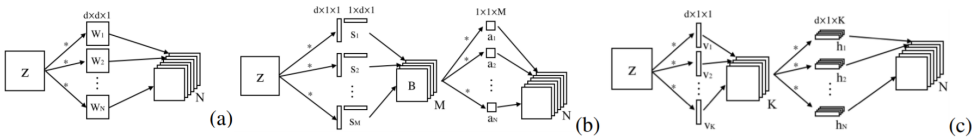


Figure 6: original source: Jaderberg et al. [85] - Low Rank Expansion Methods: (a) standard CNN filter, (b) LR approximation along the spatial dimension of 2d seperable filters and (c) extending to 3D filters where each conv. layer is factored as a sequence ot two standard conv. layers but with rectangular filters.

4.3 Applications of Tensor Decompositions to Convolutional Layers

4.3.1 Filter Decompositions

Rigamonti et al. [152] reduce computation in CNNs by learning a linear combination of separable filters, while maintaining performance.

For N 2-d filters $\{^j\}_{1\leq j\leq N}$, one can obtain a shared set of separable (rank-1) filters by minimizing the objective in Equation 49

$$\underset{\{f^j\},\{m_i^j\}}{\operatorname{argmin}} \sum_i \left(||x_i - \sum_{j=1}^{N} f^j * m_i^j||_2^2 + \lambda \sum_{j=1}^{N} ||m_i^j||_1 \right) \tag{49}$$

where $\mathbf{x}_i$ is an input image, $*$ denotes the convolution product operator, $\{m_i^j\}_{j=1...N}$ are the feature maps obtained during training and λ_1 is a regularization coefficient. This can be optimized using stochastic gradient descent (SGD) to optimize for m_i^j latent features and f^j filters.

In the first approach they identify low-rank filters using the objective in Equation 50 to penalize high-rank filters.

$$\text{argmin}_{\{s^j\},\{m_i^j\}} \sum_i \left(||x_i - \sum_{j=1}^{N} s_j * m_j^i||_2^2 + \lambda_1 \sum_{j=1}^{N} ||m_i^j||_1 + \lambda_* \sum_{j=1}^{N} ||s^j||_* \right) \tag{50}$$

where the s_j are the learned linear filters, $||*||$ is the sum of singular values (convex relaxation of the rank), and λ_* is an additional regularization parameter. The second approach involves separating the optimization of squared difference between the original filter f^i and the weighted combination of learned linear filters $w_k^j s_k$, and the sum of singular values of learned filters s.

$$\text{argmin}_{\{s_k\},\{w_k^j\}} \sum_j \left(||f^i - \sum_{k=1}^{M} w_k^j s_k||_2^2 + \lambda_* \sum_{k=1}^{M} ||s^j||_* \right) \tag{51}$$

They find empirically that decoupling the computation of the non-separable filters from that of the separable ones leads to better results compared to jointly optimizing over s^j, m_i^j and w_k^j which is a difficult optimization problem.

4.3.2 Channel-wise Decompositions

Jaderberg et al. [85] propose to approximate filters in convolutional layers using a low-rank basis of filters that have good separability in the spatial filter dimensions but make the contribution of removing redundancy across channels by performing channel-wise low-rank (LR) decompositions (LRD), leading to further speedups. This approach showed significant 2.5x speed ups and maintained performance on character recognition leading to SoTA on standard benchmarks.

4.3.3 Combining Filter and Channel Decompositions

Yu et al. [201] argue that sparse and low rank decompositions (LRDs) of weight filters should be combined as filters often exhibit both and ignoring either sparsity or LRDs requires iterative retraining and lower comrpession rates. Feature maps are reconstructed using fast-SVD. This approach allowed accuracy to be maintained for higher compression rates in few retraining steps when compared to single approaches (e.g pruning) for AlexNet, VGG-16 (15 time reduction) and LeNet CNN architectures.

5 Knowledge Distillation

Knowledge distillation (also known as knowledge distillation) involves learning a smaller network from a large network using supervision from the larger network and minimizing the entropy, distance or divergence between their probabilistic estimates.

To our knowledge, Buciluǎ et al. [16] first explored the idea of reducing model size by learning a student network from an ensemble of models. They use a teacher network to label a large amount of unlabeled data and train a student network using supervision from the pseudo labels provided by the teacher. They find performance is close to the original ensemble with 1000 times smaller network.

Hinton et al. [72] a neural network knowledge distillation approach where a relatively small model (2-hidden layer with 800 hidden units and ReLU activations) is trained using supervision (class probability outputs) for the original "teacher" model (2-hidden layer, 1200 hidden units). They showed that learning from the larger network outperformed the smaller network learning from scratch in the standard supervised classification setup. In the case of learning from ensemble, the average class probability is used as the target.

The cross entropy loss is used between the class probability outputs of the student output y^S and one-hot target y and a second term is used to ensure that the student representation z^s is similar to the teacher output z^T. This is expressed in terms of KL divergence as,

$$\mathcal{L}_{\text{KD}} = (1-\alpha)\mathbb{H}(y, y^S) + \alpha\rho^2\mathbb{H}\bigg(\phi\Big(\frac{z^T}{\rho}\Big), \phi\Big(\frac{z^S}{\rho}\Big)\bigg) \tag{52}$$

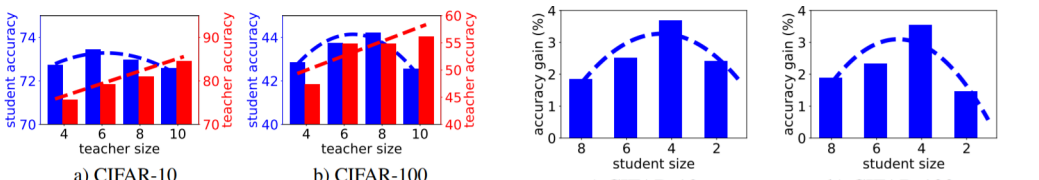
where ρ is the temperature, α balances between both terms, and ϕ represents the softmax function. The $\mathbb{H}\big(\phi(\frac{z^T}{\rho}), \phi(\frac{z^S}{\rho})\big)$ is further decomposed into $D_{\text{KL}}\big(\phi(\frac{z^T}{\rho})|\phi(\frac{z^S}{\rho})\big)$ and a constant entropy $\mathbb{H}\big(\phi(\frac{z^T}{\rho})\big)$.

The idea of training a student network on the logit outputs (i.e log of the predicted probabilities) of the teacher to gain more information from the teacher network can be attributed to the work of Ba and Caruana [7]. By using logits, as opposed to a softmax normalization across class probabilities for example, the student network better learns the relationship between each class on a log-scale which is more forgiving than the softmax when the differences in probabilities are large.

5.1 Analysis of Knowledge Distillation

The works in this subsection provide insight into the relationship between the student and teacher networks for various tasks, teacher size and network size. We also discuss work that focuses on what is required to train a well-performing student network e.g use of early stopping [174] and avoiding training the teacher network with label smoothing [132].

Theories of Why Knowledge Distillation Works For a distilled linear classifier, Phuong and Lampert [145] prove a generalization bound that shows the fast convergence of the expected loss. In the case where the number of samples is less than the dimensionality of the feature space, the weights learned by the student network are projections of the weights in the student network onto the data span. Since gradient descent makes updates that are within the data space, the student network is bounded in this space and therefore it is the best student network can approximate of the teacher network weights w.r.t the Euclidean norm. From this proof, they identify 3 important factors that contribute that explain the success of knowledge distillation - (1) the geometry of the data distribution that makes up the separation between classes greatly effects the student networks convergence rate



Distillation performance with increasing teacher size. The number of convolutional layers in student is 2.

Percentage of distilled student performance increase over the performance when it learns from scratch with varying student size. The teacher has 10 layers.

Figure 7: original source Mirzadeh et al. [126]

(2), gradient descent is biased towards a desirable minimum in the distillation objective and (3) the loss monotonically decreases proportional to size of the training set.

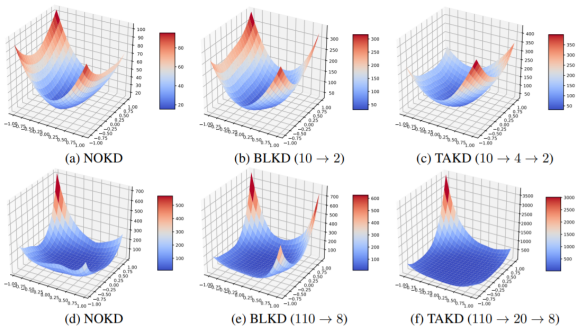
Teacher Assistant Knowledge Distillation Mirzadeh et al. [126] show that the performance of the student network degrades when the gap between the teacher and the student is too large for the student to learn from. Hence, they propose an intermediate 'teaching assistant' network to supervise and distil the student network where the intermediate networks is distilled from the teacher network.

Figure 7 shows their plot, where on the left side a) and b) we see that as the gap between the student and teacher networks widen when the student network size is fixed, the performance of student network gradually degrades. Similarly, on the right hand side, a similar trend is observed when the student network size is increased with a fixed teacher network.

Theoretical analysis and extensive experiments on CIFAR-10,100 and ImageNet datasets and on CNN and ResNet architectures substantiate the effectiveness of our proposed approach.

Their Figure 8 shows the loss surface of CNNs trained on CIFAR-100 for 3 different approaches: (1) no distillation, (2) standard knowledge distillation and (3) teaching assisted knowledge distillation. As shown, the teaching assisted knowledge distillation has a smoother surface around the local minima, corresponding to more robustness when the inputs are perturbed and better generalization.

On the Efficacy of Knowledge Distillation Cho and Hariharan [26] analyse what are some of the main factors in successfully using a teacher network to distil a student network. Their main finding is that when the gap between the student and teacher networks capacity is too large, distilling a student network that maintains performance or close to the teacher is either unattainable or difficult. They also find that the student network can perform better if early stopping is used for the teacher network, as opposed to training the teacher network to convergence.



Loss landscape around local minima.
. Top) plain CNN for student of size 2. Bottom: ResNet for student of size 8.

Figure 8: original source: Mirzadeh et al. [126]

Figure 9 shows that teachers (DenseNet and WideResNet) trained with early stopping are better suited as supervisors for the student network (DenseNet40-12 and WideResNet16-1).

Avoid Training the Teacher Network with Label Smoothing Muller et al. [132] show that because label smoothing forces the same class sample representations to be closer to each other in the embedding space, it provides less information to student network about the boundary between each class and in turn leads to poorer generlization performance. They quantify the variation in logit predictions due to the hard targets using mutual information between the input and output logit and show that label smoothing reduces the mutual information. Hence, they draw a connection

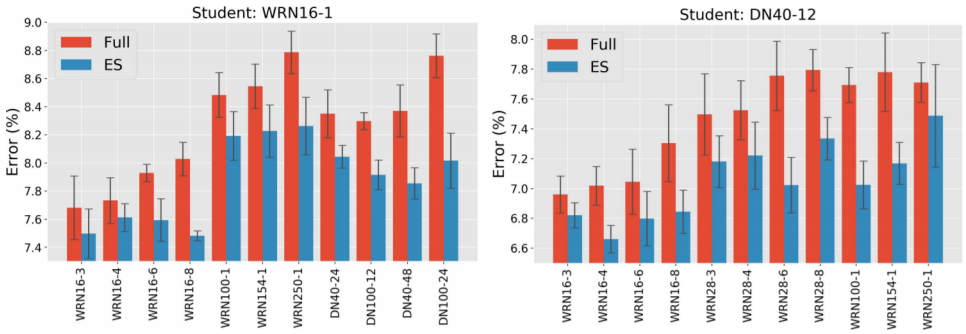


Figure 9: original source Cho and Hariharan [26]: Early Stopping on

between label smoothing and information bottleneck principle and show through experiments that label smoothing can implicitly calibrate the predictions of a DNN.

Distilling with Noisy labels Li et al. [109] distil models with noisy labels and use a small dataset with clean labels, alongside a knowledge graph that contains the label relations, to estimate risk associated with training using each noisy label. A model is trained on the clean dataset D_c and the main model is trained over the whole dataset D with noisy labels using the loss function,

$$\mathcal{L}_D(\boldsymbol{y}_i, f(\boldsymbol{x}_i)) = \lambda l(\boldsymbol{y}_i, f(\boldsymbol{x}_i)) + (1-\lambda) l(\boldsymbol{s}_i, f(\boldsymbol{x}_i)) \tag{53}$$

where $s_i = \delta[f_c^D(\boldsymbol{x}i)]$. The first loss term is cross entropy between student noisy and noisy labels and the second term is the loss between the the hard target s_i given by the model trained on clean data and the model trained on noisy data.

They also use pseudo labels $\hat{y}\lambda_i = \lambda \boldsymbol{y}_i + (1-\lambda)\boldsymbol{s}_i$ that combine noisy label $\boldsymbol{y}_i$ with the output $\boldsymbol{s}_i$ trained on D_c. This motivated by the fact that both noisy label and the predicted labels from clean data are independent and this can be closer to true labels y_i^* under conditions which they further detail in the paper.

To avoid the model trained on D_c overfitting, they assign label confidence score based on related labels from a knowledge graph, resulting in a reduction in model variance during knowledge distillation.

Distillation of Hidden Layer Activation Boundaries Instead of transferring the outputs of the teacher network, Heo et al. [70] transfer activation boundaries, essentially outputs which neurons are activated and those that are not. They use an activation loss that minimizes the difference between the student and teacher network activation boundaries, unlike previous work that focuses on the activation magnitude. Since gradient descent updates cannot be used on the non-differentiable loss, they propose an approximation of the activation transfer loss that can be minimized using gradient descent. The objective is given as,

$$\mathcal{L}(I) = ||\rho(\mathcal{T}(\mathcal{I}))\sigma\big(\mu\mathbf{1} - r(\mathcal{S}(\mathcal{I}))\big) + \big(1-\rho(\mathcal{T}(\mathcal{I}))\big) \circ \sigma(\mu\mathbf{1} + r(\mathcal{S}(\mathcal{I})))||_2^2 \tag{54}$$

where $\mathcal{S}(\mathcal{I})$ and $\mathcal{T}(\mathcal{I})$ are the neuron response tensors for student and teacher networks, $\rho(\mathcal{T}(I))$ is the the activation of teacher neurons corresponding to class labels, $r(\mathcal{S}(\mathcal{I}))$ is the , r is a connector function (a fully connected layer in their experiments) that converts a neuron response vector of student to the same size as the teacher vector, $\circ$ is elementwise product of vectors and μ is the margin to stabilize training.

Data-Free Knowledge Distillation Lopes et al. [119] try distill in the scenario where it is not possible to have access to the original data the teacher network was trained on. This can occur due to privacy issues (e.g personal medical data, models trained case-based legal data) or the data is no longer available or some way corrupted. They store the sufficient statistics (e.g mean and covariance) of activation outputs from the original data along with the pretrained teacher network to reconstruct the original training data input. This is achieved by trying to find images that have the highest

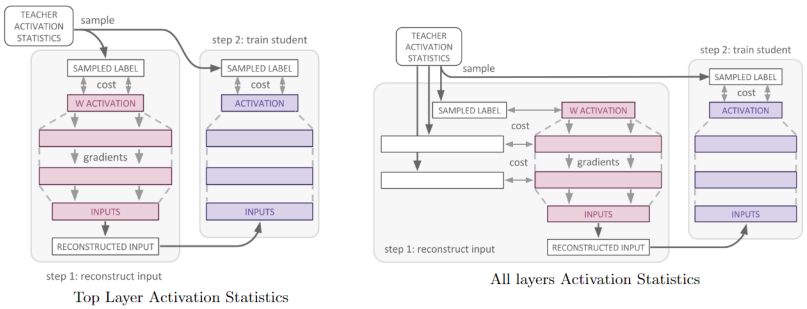


Figure 10: original source Lopes et al. [119]: Data Free Knowledge Distillation. The left shows

representational similarity to those given by the representations from the activation records of the teacher network. Gaussian noise is passed as input to the teacher and update gradients to the noise to minimize the difference between the recorded activation outputs and those of the noisy image and repeat this to reconstruct the teachers view of the original data.

The left figure in Figure 10 shows the activation statistics for the top layer and a sample drawn that is used to optimize the input to teacher network to reconstruct the activations. The reconstructed input is then fed to the student network. On the right, the same procedure follows but for reconstructing activations for all layers of the teacher network.

They manage to compress the teacher network using the reconstructed inputs to half the size in the student network, only from using the metadata. The amount of compression achieved is contingent on the quality of the metadata, in their case they only used activation statistics.

5.2 Distilling Recurrent (Autoregressive) Neural Networks

Although the work by Buciluǎ et al. [16] and Hinton et al. [72] has often proven successful for reducing the size of neural models in other non-sequential tasks, many sequential tasks in NLP and CV have high-dimensional outputs (machine translation, pixel generation, image captioning etc.). This means using the teachers probabilistic outputs as targets can be expensive.

Kim and Rush [94] use the teachers hard targets (also 1-hot vectors) given by the highest scoring beam search prediction from an encoder-decoder RNN, instead of the soft output probability distribution. The teacher distribution $q(y_t|x)$ is approximated by its mode:$q(y_s|x) \approx 1t = \text{argmax}_{y_t \in \mathcal{Y}} q(y_t|x)$ with the following objective

$$\mathcal{L}_{SEQ-MD} = -\mathbb{E}_{x\sim D} \sum_{y_t \in \mathcal{Y}} p(y_t|x) \log p(y_t|x) \approx -\mathbb{E}_{x\sim D}, \hat{y}_s = \underset{y_t \in \mathcal{Y}}{\text{argmax}}\, q(y_t|x)[\log p(y_t = \hat{y}_s|x)] \tag{55}$$

where $y_t \in \mathcal{Y}$ are teacher targets (originally defined by the predictions with the highest scoring beam search) in the space of possible target sequences. When the temperature $\tau \to 0$, this is equivalent to standard knowledge distillation.

In sequence-level interpolation, the targets from the teacher with the highest *similarity* with the ground truth are used as the targets for the student network. Experiments on NMT showed performance improvements compared to soft targets and further pruning the distilled model results in a pruned student that has 13 times fewer parameters than the teacher network with a 0.4 decrease in BLEU metric.

Noisy Student Training Sau and Balasubramanian [159] propose to use noise to simulate learning from multiple teacher networks by simply adding Gaussian noise the logit outputs of the teacher network, resulting in better compression when compared to training with the original logits as targets for the teacher network. They choose a set of samples from each mini-batch with a probability α to perturbed by noise while the remaining samples are unchanged. They find that a relatively high $alpha = 0.8$ performed the best for image classification task, corresponding to 80% of teacher logits having noise. Park et al. [140] have extended this idea to speech recognition and similarly found

high compression rates using a distilled DNN. Related to noisy student training, Han et al. [61] have pointed out proposed co-teaching where two networks learn from each other where one has clean outputs and the other has noisy outputs. This avoids a single DNN from learning to memorize the noisy labels and select samples from each mini-batch that the networks should learn from and avoid those samples which correspond to noisy labels. Since both networks have different ways of learning, they filter different types of error occurring from the noisy labels and this information is communicated mutually. This strategy could also be useful for using the teacher network to provide samples to a smaller student network that improve the learning of the student.

5.3 Distilling Transformer-based (Non-Autoregressive) Networks

Knowledge distillation has also been applied to very large transformer networks, predominantly on BERT [41] given its wide success in NLP. Thus, there has been a lot of recent work towards reducing the size of BERT and related models using knowledge distillation.

DistilBERT Sanh et al. [158] achieves distillation by training a smaller BERT on very large batches using gradient accumulation, uses dynamic masking, initializes the student weights with teacher weights and removes the next sentence prediction objective. They train the smaller BERT model on the original data BERT was trained on and fine that DistilBERT is within 3% of the original BERT accuracy while being 60% faster when evaluated on the GLUE [184] benchmark dataset.

BERT Patient Knowledge Distillation Instead of minimizing the soft probabilities between the student and teacher network outputs, Sun et al. [171] propose to also learn from the intermediate layers of the BERT teacher network by minimizing the mean squared error between adjacent and normalized hidden states. This loss is combined with the original objective proposed by Hinton et al. [72] which showed further improves in distilling BERT on the GLUE benchmark datasets [184].

TinyBERT TinyBERT [87] combines multiple Mean Squared Error (MSE) losses between embeddings, hidden layers, attention layers and prediction outputs between S and T. The TinyBERT distillation objective is shown below, where it combines multiple reconstruction errors between S and T embeddings (when m=0), between the hidden and attention layers of S and T when $M \geq m > 0$ where M is index of the last hidden layer before prediction layer and lastly the cross entropy between the predictions where t is the temperature of the softmax.

$$L_{layer}\big(S_m, T_g(m)\big) = \begin{cases} \text{MSE}(\mathbf{E}^S \mathbf{W}_e \mathbf{E}^T) & m = 0 \\ \text{MSE}(\mathbf{H}^S \mathbf{W}_h, \mathbf{H}^T) + \frac{1}{h}\sum_{i=1}^{h} \text{MSE}(\mathbf{A}_i^S, \mathbf{A}_i^T) & M \geq m > 0 \\ \text{softmax}(\boldsymbol{z}^T) \cdot \text{log-softmax}(\boldsymbol{z}^S/t) & m = M+1 \end{cases}$$

Through many ablations in experimentations, they find distilling the knowledge from multi-head attention layers to be an important step in improving distillation performance.

ALBERT Lan et al. [100] proposed factorized embeddings to reduce the size of the vocabulary embeddings and parameter sharing across layers to reduce the number of parameters without a performance drop and further improve performance by replacing next sentence prediction with an inter-sentence coherence loss. ALBERT is. 5.5% the size of original BERT and has produced state of the art results on top NLP benchmarks such as GLUE [184], SQuAD [149] and RACE [98].

BERT Distillation for Text Generation Chen et al. [25] use a conditional masked language model that enables BERT to be used on generation tasks. The outputs of a pretrained BERT teacher network are used to provide sequence-level supervision to improve Seq2Seq model and allow them to plan ahead. Figure 11 illustrates the process, showing where the predicted probability distribution for the remaining tokens is minimized with respect to the masked output sequence from the BERT teacher.

Applications to Machine Translation Zhou et al. [208] seek to better understand why knowledge distillation leads to better non-autoregressive distilled models for machine translation. They find that the student network finds it easier to model variations in the output data since the teacher network reduces the complexity of the dataset.

5.4 Ensemble-based Knowledge Distillation

Ensembles of Teacher Networks for Speech Recognition Chebotar and Waters [23] use the labels from an ensemble of teacher networks to supervise a student network trained for acoustic modelling. To choose a good ensemble, one can select an ensemble where each individual model potentially make different errors but together they provide the student with strong signal for learning. Boosting weights each sample based proportional to its misclassification rate. Similarly this can used on the ensemble to learn which outputs from each model to use for supervision. Instead of learning from a combination of teachers that are best by using an oracle that approximates the best outcome of the ensemble for automatic speech recognition (ASR) as

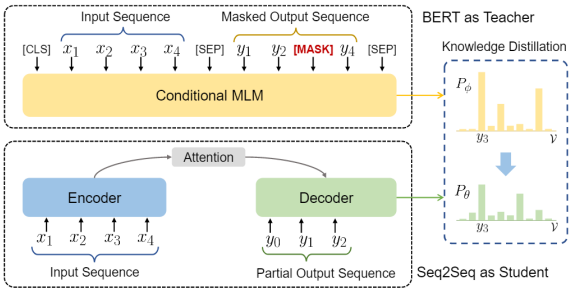


Illustration of distilling knowledge from BERT for text generation. See Section 3.2 and 3.3

Figure 11: original source [25]: BERT Distillation for Text Generation

$$P_{\text{oracle}}(s|x) = \sum_{i=1}^{N} [O(u) = i] P_i(s|x) = P_O(u)(s|x) \tag{56}$$

where the oracle $O(u) \in 1 \dots N$ that contains N teachers assigns all the weight to the model that has the lowest word errors for a given utterance u. Each model is an RNN of different architecture trained with different objectives and the student s is trained using the Kullbeck Leibler (KL) divergence between oracle assigned teachers output and the student network output. They achieve an 8.9% word error rate improvement over similarly structured baseline models.

Freitag et al. [49] apply knowledge distillation to NMT by distilling an ensemble of networks and oracle BLEU teacher network into a single NMT system. The find a student network of equal size to the teacher network outperforms the teacher after training. They also reduce training time by only updating the student networks with filtered samples based on the knowledge of the teacher network which further improves translation performance.

Cui et al. [32] propose two strategies for learning from an ensemble of teacher network; (1) alternate between each teacher in the ensemble when assigning labels for each mini-batch and (2) simultaneously learn from multiple teacher distributions via data augmentation. They experiment on both approaches where the teacher networks are deep VGG and LSTM networks from acoustic models.

Cui et al. [32] extend knowledge distillation to multilingual problems. They use multiple pretrained teacher LSTMs trained on multiple low-resource languages to distil into a smaller standard (fully-connected) DNN. They find that student networks with good input features makes it easier to learn from the teachers labels and can improve over the original teacher network. Moreover, from their experiments they suggest that allowing the ensemble of teachers learn from one another, the distilled model further improves.

Mean Teacher Networks Tarvainen and Valpola [174] find that averaging the models weights of an ensemble at each epoch is more effective than averaging label predictions for semi-supervised learning. This means the Mean Teacher can be used as unsupervised learning distillation approach as the distiller does not need labels. than methods which rely on supervision for each ensemble model. They find this straightforward approach to outperform previous ensemble based distillation approaches [99] when only given 1000 labels on the Street View House View Number [SVHN; 55] dataset. Moreover, using Mean Teacher networks with Residual Networks achieved SoTA with 4000 labels from 10.55% error to 6.28% error.

on-the-fly native ensemble Zhu et al. [212] focus on using distillation on the fly in a scenario where the teacher may not be fully pretrained or it does not have a high capacity. This reduces

compression from a two-phase (pretrain then distil) to one phase where both student and teacher network learn together. They propose an On the fly Native Ensemble (ONE) learning strategy that essentially learns a strong teacher network that assists the student network as it is learning. Performance improvements for on the fly distillation are found on the top benchmark image classification datasets.

Multi-Task Teacher Networks Liu et al. [116] perform knowledge distillation for performing multi-task learning (MTL), using the outputs of teacher models from each natural language understanding (NLU) task as supervision for the student network to perform MTL. The distilled MT-DNN outperforms the original network on 7 out of 9 NLU tasks (includes sentence classification, pairwise sentence classification and pairwise ranking) on the GLUE [184] benchmark dataset.

5.5 Reinforcement Learning Based Knowledge Distillation

Knowledge distillation has also been performed using reinforcement learning (RL) where the objective is to optimize for accumulated of rewards where the reward function can be task-specific. Since not all problems optimize for the log-likelihood, standard supervised learning can be a poor surrogate, hence RL-based distillation can directly optimize for the metric used for evaluaion.

Network2Network Compression Ashok et al. [6] propose Network to Network (N2N) compression in policy gradient-based models using a RNN policy network that removes layers from the 'teacher' model while another RNN policy network then reduces the size of the remaining layers. The resulting policy network is trained to find a locally optimal student network and accuracy is considered the reward signal. The policy networks gradients are updated accordingly, achieving a compression ratio of 10 for ResNet-34 while maintaining similar performance to the original teacher network.

FitNets Romero et al. [153] propose a student network that has deeper yet smaller hidden layers compared to the teacher network. They also constrain the hidden representations between the networks to be similar. Since the hidden layer size for student and teacher will be different, they project the student layer to into an embedding space of fixed size so that both teacher and student hidden representations are of the same size. Equation 57 represents the Fitnet loss where the first term represents the cross-entropy between the target y_{true} and the student probability P_S, while $H(P_T^\tau, P_S^\tau)$ represents the cross entropy between the normalized and flattened teachers hidden representation P_T^τ and the normalized student hidden representation P_S^τ where γ controls the influence of this similarity constraint.

$$\mathcal{L}_{\text{MD}}(\mathbf{W}_S) = H(y_{\text{true}}, P_S) + \gamma H(P_T^\tau, P_S^\tau) \tag{57}$$

Equation 58 shows the loss between the teacher weights $\mathbf{W}_{Guided}$ for a given layer and the reconstructed weights $\mathbf{W}_r$ which are the weights of a corresponding student network projected using a convolutional layer (cuts down computation compared to a fully-connected projection layer) to the same hidden size of the teacher network weights.

$$\mathcal{L}_{\text{HT}}(\mathbf{W}_T, \mathbf{W}_r) = \frac{1}{2}||u_h(\boldsymbol{x}; \mathbf{W}_{\text{Hint}}) - r(v_g(\boldsymbol{x}; \mathbf{W}_{\text{T}}); \mathbf{W}_r)||^2 \tag{58}$$

where u_h and v_g are the teacher/student deep nested functions up to their respective hint/guided layers with parameters $\mathbf{W}_{\text{Hint}}$ and $\mathbf{W}_{\text{Guided}}$, r is the regressor function on top of the guided layer with parameters $\mathbf{W}_r$. Note that the outputs of uh and r have to be comparable, i.e., u_h and r must be the same non-linearity. The teacher tries to imitate the flow matrices from the teacher which are defined as the inner product between feature maps, such as layers in a residual block.

5.6 Generative Modelling Based Knowledge Distillation

Here, we describe how two commonly used generative models, variational inference (VI) and generative adversarial networks (GANs), have been applied to learning a student networks.

5.6.1 Variational Inference Learned Student

Hegde et al. [69] propose a variational student whereby VI is used for knowledge distillation. The parameters induced by using VI-based least squares objective are sparse, improving the generalizability of the student network. Sparse Variational Dropout (SVD) Kingma et al. [95], Molchanov et al. [129] techniques can also be used in this framework to promote sparsity in the network. The VI objective is shown in Equation 59, where $\boldsymbol{z}^s$ and $\boldsymbol{z}^t$ are the output logits from student and teacher networks.

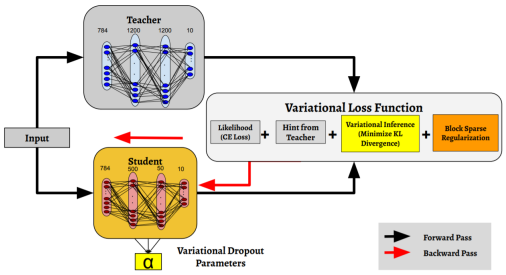


Figure 12: Variational Student Framework (original source: Hegde et al. [69])

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \mathbf{W}_s, \mathbf{W}_t, \alpha) = -\frac{1}{N}\sum_{n=1}^{N} \boldsymbol{y}_n \log(\boldsymbol{z}_n^s) + \lambda_T \left[2T^2 D_{\text{KL}}\left(\sigma'\left(\frac{\boldsymbol{z}^s}{T}\right) \middle|\middle| \sigma'\left(\frac{\boldsymbol{z}^t}{T}\right)\right)\right] + \lambda_V \mathcal{L}_{\text{KL}}(\mathbf{W}_s, \alpha) + \lambda_g \sum_{m=1}^{M} \left| \max_{n,k,h,l} W_{T:S}(m,n,k,h,l)\right| \quad (59)$$

Figure 12 shows their training procedure and loss function that consist of the learning compact and sparse student networks. The roles of different terms in variational loss function are: likelihood - for independent student network's learning; hint - learning induced from teacher network; variational term - promotes sparsity by optimizing variational dropout parameters, α; Block Sparse Regularization - promotes and transfers sparsity from the teacher network.

5.6.2 Generative Adversarial Student

GANs train a discriminator binary classifier f_w to discriminate between real samples x and generated samples $g_\theta(z)$ that are given by a generator network g_θ and z is sampled from p_g a known distribution such as a Gaussian. A minimax objective is used to minimize the misclassifications of the discriminator while maximizing the generators accuracy of tricking the discriminator. This is formulated as,

$$\min_{\theta\in\Theta}\max_{\boldsymbol{w}\in W} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log(f_{\boldsymbol{w}}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - f_{\boldsymbol{w}}(g_\theta(\boldsymbol{z})))] \quad (60)$$

where the global minimum is found when the generator distribution p_g is similar to the data distribution p_{data} (referred to as the nash equilibrium).

Naive GAN & Knowledge Distilled GAN Wang et al. [186] learn a Generative Adversarial Student Network where the generator learns from the teacher network using the minimax objective in Equation 60. They reduce the variance in gradient updates which leads less epochs requires to train to convergence, by using the Gumbel-Max trick in the formulation of GAN knowledge distillation.

First they propose Naive GAN (NaGAN) which consists of a classifier C and a discriminator D where C generates pseudo labels given a sample x from a categorical distribution $p_c(\boldsymbol{y}|\boldsymbol{x})$ and D distinguishes between the true targets and the generated ones. The objective for NaGAN is express as,

$$\min_{c}\max_{d} V(c,d) = \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d(\boldsymbol{x},\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p_d^o(\boldsymbol{x},\boldsymbol{y}))] \quad (61)$$

where $V(c,d)$ is the value function. The scoring functions of C and D are $h(\boldsymbol{x}, y)$ and $g(x, y)$ respectively. Then $p_c(y|\boldsymbol{x})$ and $p_d^o(\boldsymbol{x}, y)$ are expressed as,

$$p_c(y|\boldsymbol{x}) = \phi(h(\boldsymbol{x}, y)), \quad p_d^o(\boldsymbol{x}, y) = \sigma(g(\boldsymbol{x}, y)) \quad (62)$$

where ϕ is the softmax function and σ is the sigmoid function. However, NaGAN requires a large number of samples and epochs to converge to nash equilibrium using this objective, since the gradients from D that update C can often vanish or explode.

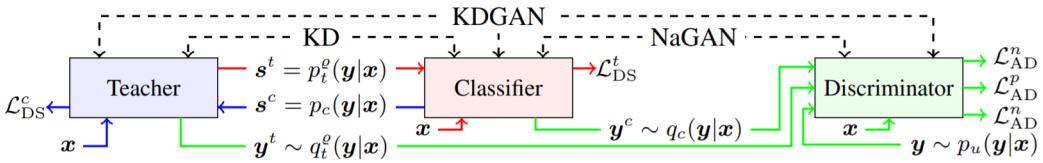


Figure 13: original source Wang et al. [186]: Comparison among KD, NaGAN, and KDGAN

This brings us to their main contribution, Knowledge Distilled GAN (KDGAN).

KDGAN somewhat remedy the aforementioned converegence problem by introducing a pretrained teacher network T along with C and D. The objective then consists of a distillation ℓ_2 loss component between T and C and adversarial loss between T and D. Therfore, both C and T aim to fool D by generating fake labels that seem real, while C tries to distil the knowledge from T such that both C and T agree on a good fake label.

The student network convergence is tracked by observing the generator outputs and loss changes. Since the gradient from T tend to have low variance, this can help C converge faster, reaching a nash equilibrium. The difference between these models is illustrated in Figure 13.

Compressing Generative Adversarial Networks Aguinaldo et al. [2] compress GANs achieving high compression ratios (58:1 on CIFAR-10 and 87:1 CelebA) while maintaining high Inception Score (IS) and low Frechet Inception Distance (FID). They're main finding is that a compressed GAN can outperform the original overparameterized teacher GAN, providing further evidence for the benefit of compression in very larrge networks. Figure 14 illustrates the student-teacher training using a joint loss between the student GAN discriminator and teacher generator DCGAN.

Student-teacher training framework with joint loss for student training. The teacher generator was trained using deconvolutional GAN [DCGAN; 147] framework.

They use a joint training loss to optimize that can be expressed as,

$$\min_{\theta\in\Theta}\max_{\boldsymbol{w}\in W}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log(f_{\boldsymbol{w}}(\boldsymbol{x})]+\mathbb{E}_{z\sim p_z}\left[\alpha\log(1-f_{\boldsymbol{w}}(g_\theta(z)))+(1-\alpha)g_{\text{teacher}}||(z)-g_\theta(z)||^2\right] \quad (63)$$

where α controls the influence of the MSE loss between the logit predictions $g_{\text{teacher}}(z)$ and $g_\theta(z)$ of teacher and student respectively. The terms with expectations correspond to the standard adversarial loss.

5.7 Pairwise-based Knowledge Distillation

Apart from pointwise classification tasks, knowledge distillation has also been performed for pairwise tasks.

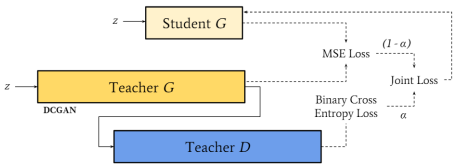


Figure 14: original source Aguinaldo et al. [2]: Student Teacher GAN Training

Similarity-preserving Knowledge Distillation Semantically similar inputs tend to have similar activation patterns. Based on this premise, Tung and Mori [179] have propose knowledge distillation such that input pair similarity scores from the student network are similar to those from the teacher network. This can be a pairwise learning extension of the standard knowledge distillation approaches.

THey aim to preserve similarity between student and pretrained teacher activations for given batch of similar and dissimilar input pairs.

For a batch b, a similarity matrix $G(l^{'})_S \in \mathbb{R}^{b\times b}$ is produced between their student activations $A_S^{(l^{'})}$ at the $l^{'}$ layer and teacher activations $A_T^{(l)}$ at the l-th layer. The objective is then defined as the cross

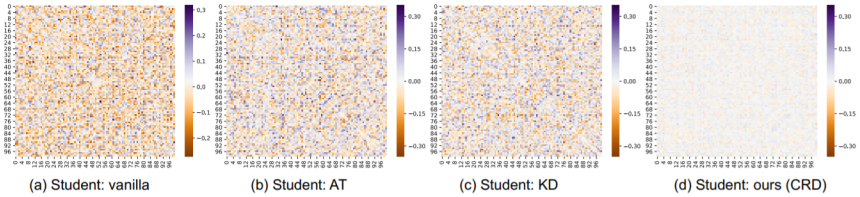


Figure 15: original source: Tian et al. [176]

entropy between the student logit output $\sigma(\boldsymbol{z}_s)$ and target y summed with the similarity preserving distillation loss component on the RHS of Equation 64,

$$\mathcal{L} = \mathcal{L}_{\text{ce}}(\boldsymbol{y}, \phi(\boldsymbol{Z}_S)) + \frac{\gamma}{b^2} \sum_{(l,l') \in \mathcal{I}} ||\mathbf{G}_T^{(l)} - \mathbf{G}_S^{(l')}||_F^2 \tag{64}$$

where $||\cdot||_F$ denotes the Frobenius norm, $\mathcal{I}$ is the total number of layer pairs considered and γ controls the influence of similarity preserving term between both networks.

In the transfer learning setting, their experiments show that similarity preserving can be a robust way to deal with domain shift. Moreover, this method complements the SoTA attention transfer [202] approach.

Contrastive Representation Distillation Instead of minimizing the KL divergence between the scalar outputs of teacher network T and student network S, Tian et al. [176] propose to preserve structural information of the embedding space. Similar to Hinton et al. [73], they force the representations between the student and teacher network to be similar but instead use a constrastive loss that moves positive paired representations closer together while positive-negative pairs away. This contrastive objective is given by,

$$\begin{aligned} f^{S*} = \underset{f^S}{\text{argmax}} \max_h \mathcal{L}_{\text{critic}}(h) = \\ \underset{f^S}{\text{argmax}} \max_h \mathbb{E}_q(T, S|C=1)[\log h(T,S)] + N\mathbb{E}_q(T,S|C=0)[\log(1-h(T,S))] \end{aligned} \tag{65}$$

where $h(T,S) = \frac{e^{g^T(T)' g^S(S)'/\tau}}{e^{g^T(T) g^S(S)/\tau} + NM}$, M is number of data samples, τ is the temperature. If the dimensionality of the outputs from g^T and g^S are not equal, a linear transformation is made to fixed size followed by an ℓ_2 normalization.

Figure 15 demonstrates how the correlations between student and teacher network are accounted for in CRD (d) while in standard teacher-student networks (a) ignores the correlations and to a less extent this is also found for attention transfer (b) [203] and the student network distilled by KL divergence (c) [72].

Relational Knowledge Distillation Park et al. [143] apply knowledge distillation to relational data and propose distance (huber) and angular-based (cosine proximity) loss functions that account for different relational structures and claim that metric learning allows the student relational network to outperform the teacher network on achieving SoTA on relational datasets.

The $\psi(\cdot)$ similarity function from the relation teacher network outputs a score that is transferred to as a pseudo target for the teacher network to learn from as,

$$\delta(x,y) = \begin{cases} \frac{1}{2}\sum_{i=1}^{N}(x-y)^2 & \text{for} \quad |x-y| \leq 1 \\ |x-y| - 1 & \text{otherwise} \end{cases}$$

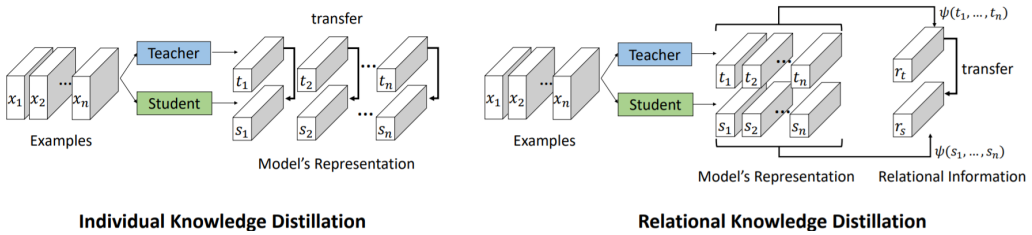


Figure 16: original source Park et al. [143]: Individual knowledge distillation (IKD) vs. relational knowledge distillation (RKD)

In the case of the angular loss shown in Equation 66, $\boldsymbol{e}^{ij} = \frac{t_i - t_j}{||t_i - t_j||_2}$, $e^{kj} = \frac{t_k - t_j}{||t_k - t_j||^2}$.

$$\psi_A(t_i, t_j, t_k) = \cos \angle t_i t_j t_k = \langle \boldsymbol{e}_{ij}, \boldsymbol{e}_{kj} \rangle \tag{66}$$

They find that measuring the angle between teacher and student outputs as input to the huber loss $\mathcal{L}_delta$ leads to improved performance when compared to previous SoTA on metric learning tasks.

$$\mathcal{L}_{rmd-a} = \sum_{(x_i, x_j, x_k) \in \mathcal{X}^3} l_\delta \psi_A(t_i, t_j, t_k), \psi_A(s_i, s_j, s_k) \tag{67}$$

This is then used as a regularization terms to the task specific loss as,

$$\mathcal{L}_{\text{task}} + \lambda_{\text{MD}} \mathcal{L}_{MD} \tag{68}$$

When used in metric learning the triplet loss shown in Equation 69 is used.

$$\mathcal{L}_{\text{triplet}} = \left[||f(\boldsymbol{x}_a) - f(\boldsymbol{x}_p)||_2^2 - ||f(\boldsymbol{x}_a) - f(\boldsymbol{x}_n)||_2^2 + m \right]_+ \tag{69}$$

Figure 17 shows the test data recall@1 on tested relational datasets. The teacher network is trained with the triplet loss and student distils the knowledge using Equation 68. Left of the dashed line are results on the training domain while on the right shows results on the remaining domains.

Song et al. [169] use attention-based knowledge distillation for fashion matching that jointly learns to match clothing items while incorporating domain knowledge rules defined by clothing description where the attention learns to assign weights corresponding to the rule confidence.

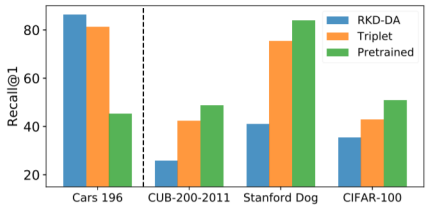


Figure 17: original source: [143]

6 Quantization

Quantization is the process of representing values with a reduced number of bits. In neural networks, this corresponds to weights, activations and gradient values. Typically, when training on the GPU, values are stored in 32-bit floating point (FP) single precision. *Half-precision* for floating point (FP-16) and integer arithmetic (INT-16) are also commonly considered. INT-16 provides higher precision but a lower dynamic range compared to FP-16. In FP-16, the result of a multiplication is accumulated into a FP-32 followed by a down-conversion to return to FP-16.

To speed up training, faster inference and reduce bandwidth memory requirements, ongoing research has focused on training and performing inference with lower-precision networks using integer precision (IP) as low as INT-8 INT-4, INT-2 or 1 bit representations [35]. Designing such networks makes it easier to train such networks on CPUs, FPGAs, ASICs and GPUs.

Two important features of quantization are the range of values that can be represented and the bit spacing. For the range of signed integers with n bits, we represent a range of [-2n-1, 2n-2] and for full precision (FP-32) the range is $+/-3.4 \times 1038$. For signed integers, there $2n$ values in that range and approximately 4.2×109 for FP-32. FP can represent a large array of distributions which is useful for neural network computation, however this comes at larger computational costs when compared to integer values. For integers to be used to represent weight matrices and activations, a FP scale factor is often used, hence many quantization approaches involve a hybrid of mostly integer formats with FP-32 scaling numbers. This approach is often referred to as mixed-precision (MP) and different MP strategies have been used to avoid overflows during training and/or inference of low resolution networks given the limited range of integer formats.

In practice, this often requires the storage of hidden layer outputs with full-precision (or at least with represented with more bits than the lower resolution copies). The main forward-pass and backpropogation is carried out with lower resolution copies and convert back to the full-precision stored "accumulators" for the gradient updates.

In the extreme case where binary weights (-1, 1) or 2-bit ternary weights (-1, 0, 1) are used in fully-connected or convolutional layers, multiplications are not used, only additions and subtractions. For binary activations, bitwise operations are used [150] and therefore addition is not used. For example, Rastegari et al. [150] proposed XNOR-Networks, where binary operations are used in a network made up of xnor gates which approximate convolutions leading to 58 times speedup and 32 times memory savings.

6.1 Approximating High Resolution Computation

Quantizing from FP-32 to 8-bit integers with retraining can result in an unacceptable drop in performance. Retraining quantized networks has shown to be effective for maintaining accuracy in some works [58]. Other work [40] compress gradients and activations from FP-32 to 8 bit approximations to maximize bandwidth use and find that performance is maintained on MNIST, CIFAR10 and ImageNet when parallelizing both model and data.

The quantization ranges can be found using k-means quantization [118], product quantization [86] and residual quantization [17]. Fixed point quantization with optimized bit width can reduce existing networks significantly without reducing performance and even improve over the original network with retraining [110].

Courbariaux et al. [31] instead scale using shifts, eliminating the necessity of floating point operations for scaling. This involves an integer or fixed point multiplication, as apart of a dot product, followed by the shift.

Dettmers [40] have also used FP-32 scaling factors for INT-8 weights and where the scaling factor is adapted during training along with the activation output range. They also consider not adapting the min-max ranges online and clip outlying values that may occur as a a result of this in order to drastically reduce the min-max range. They find SoTA speedups for CNN parallelism, achieving a 50 time speedup over baselines on 96 GPUs.

Gupta et al. [57] show that stochastic rounding techniques are important for FP-16 DNNs to converge and maintain test accuracy compared to their FP-32 counterpart models. In stochastic rounding the weight x is rounded to the nearest target fixed point representation $[x]$ with probability $1-(x-[x])/\epsilon$ where ϵ is the smallest positive number representable in the fixed-point format, otherwise x is rounded to $x+\epsilon$. Hence, if x is close to $[x]$ then the probability is higher of being assigned $[x]$. Wang et al. [185] train DNNs with FP-8 while using FP-16 chunk-based accumulations with the aforementioned stochastic rounding hardware.

The necessity of stochastic rounding, and other requirements such as loss scaling, has been avoided using customized formats such as Brain float point [(BFP) 89] which use FP-16 with the same number of exponent bits as FP-32. Cambier et al. [18] recently propose a shifted and squeezed 8-bit FP format (S2FP-8) to also avoid the need of stochastic rounding and loss scaling, while providing dynamic ranges for gradients, weights and activations. Unlike other related 8-bit techniques [122], the first and last layer do not need to be in FP32 format, although the accmulator converts the outputs to FP32.

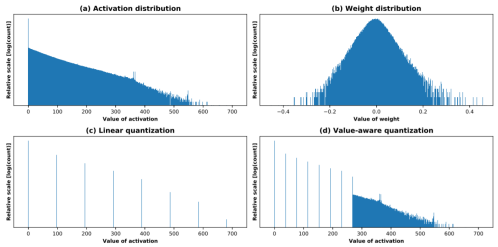
6.2 Adaptive Ranges and Clipping

Park et al. [141] exploit the fact that most the weight and activation values are scattered around a narrower region while larger values outside such region can be represented with higher precision. The distribution is demonstrated in Figure 18, which displays the weight distribution for the 2^{nd} layer in the LeNet CNN network. Instead of using linear quantization shown in (c), a smaller bit interval is used for the region of where most values lie (d), leading to less quantization errors.

They propose 3-bit activations for training quantized ResNet and Inception CNN architectures during retraining. For inference on this retrained low precision trained network, weights are also quantized to 4-bits for inference with 1% of the network being 16-bit scaling factor scalars, achieving accuracy within 1% of the original network. This was also shown to be effective in LSTM network on language modelling, achieving similar perplexities for bitwidths of 2, 3 and 4.

Migacz [125] use relative entropy to measure the loss of information between two encodings and aim minimize the KL divergence between activation output values. For each layer they store histgrams of activations, generate quantized distributions with different saturation thresholds and choose the threshold that minimizes the KL divergence between the original distribution and the quantized distribution.

Banner et al. [10] analyze the tradeoff between quantization noise and clipping distortion and derive an expression for the mean-squared error degradation due to clipping. Optimizing for this results in choosing clipping values that improve 40% accuracy over standard quantization of VGG16-BN to 4-bit integer.



Activation and weight distributions of second convolutional layer in GoogLeNet.

Figure 18: original source Park et al. [142]: Weight and Activation Distributions Before and After Quantization

Another approach is to use scaling factors per group of weights (e.g channels in the case of CNNs or internal gates in LSTMs) as opposed to whole layers, particularly useful when the variance in weight distribution between the weight groupings is relatively high.

6.3 Robustness to Quantization and Related Distortions

Merolla et al. [123] have studied the effects of different distortions on the weights and activations, including quantization, multiplicative noise (aking to Gaussian DropConnect), binarization (sign) along with other nonlinear projections and simply clipping the weights. This suggests that neural networks are robust to such distortions at the expense of longer convergence times.

In the best case of these distortions, they can achieve 11% test error on CIFAR-10 with 0.68 effective bits per weight. They find that training with weight projections other than quantization performs relatively well on ImageNet and CIFAR-10, particularly their proposed stochastic projection rule that leads to 7.64% error on CIFAR-10.

Others have also shown DNNs robustness to training binary and ternary networks [57, 31], albeit a larger number of bit weight and ternary weights that are required.

6.4 Retraining Quantized Networks

Thus far, these post-training quantization (PTQ) methods without retraining are mostly effective on overparameterized models. For smaller models that are already restricted by the degrees of freedom, PTQ can lead to relatively large performance degradation in comparison to the overparameterized regime, which has been reflected in recent findings that architectures such as MobileNet suffer when using PTQ to 8-bit integer formats and lower [84, 96].

Hence, retraining is particularly important as the number of bits used for representation decreases e.g 4-bits with range [-8, 8]. However, quantization results in discontinuities which makes differentiation during backpropogation difficult.

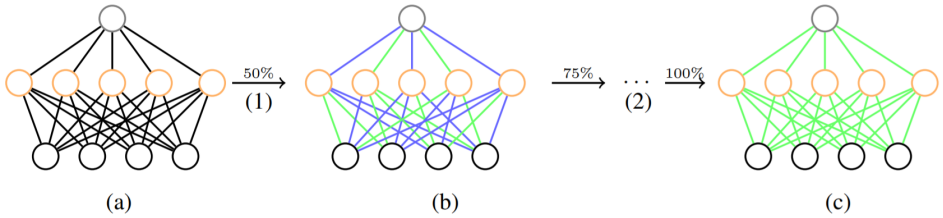


Figure 19: Quantized Knowledge Distillation (original source: [206])

To overcome this limitation, Zhou et al. [209] quantized gradients to 6-bit number and stochastically propogate back through CNN architectures such as AlexNet using straight through estimators, defined as Equation 70. Here, a real number input $r_i \in [0, 1]$ to a n-bit number output $r_o \in [0, 1]$ and $\mathcal{L}$ is the objective function.

$$\textbf{Forward} : r_o = \frac{1}{2^n - 1}\text{round}((2n - 1)r_i) \tag{70}$$

$$\textbf{Backward} : \frac{\partial \mathcal{L}}{\partial r_i} = \frac{\partial \mathcal{L}}{\partial r_o} \tag{71}$$

To compute the integer dot product of r_0 with another $n - bit$ vector, they use Equation 72, with a computational complexity of $\mathcal{O}(MK)$, directly proportional to bitwidth of x and y. Furthermore, bitwise kernels can also be used for faster training and inference

$$x \cdot y = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k}\text{bitcount}[\text{and}(c_m(\boldsymbol{x}), c_k(\boldsymbol{y}))] \tag{72}$$

$$c_m(\boldsymbol{x})i, c_k(\boldsymbol{y}) \quad i \in \{0, 1\} \quad \forall i, m, k \tag{73}$$

Model Distilled Quantization An overview of our incremental network quantization method. (a) Pre-trained full precision model used as a reference. (b) Model update with three proposed operations: weight partition, group-wise quantization (green connections) and re-training (blue connections). (c) Final low-precision model with all the weights constrained to be either powers of two or zero. In the figure, operation (1) represents a single run of (b), and operation (2) denotes the procedure of repeating operation (1) on the latest re-trained weight group until all the non-zero weights are quantized. Our method does not lead to accuracy loss when using 5-bit, 4-bit and even 3-bit approximations in network quantization. For better visualization, here we just use a 3-layer fully connected network as an illustrative example, and the newly re-trained weights are divided into two disjoint groups of the same size at each run of operation (1) except the last run which only performs quantization on the re-trained floating-point weights occupying 12.5% of the model weights.

Polino et al. [146] use a distillation loss with respect to the teacher network whose weights are quantized to set number of levels and quantized teacher trains the 'student'. They also propose differentiable quantization, which optimizes the location of quantization points through stochastic gradient descent, to better fit the behavior of the teacher model.

Quantizing Unbounded Activation Functions When the nonlinear activation unit used is not bounded in a given range, it is difficult to choose the bit range. Unlike sigmoid and tanh functions that are bounded in $[0, 1]$ and $[-1, 1]$ respectively, the ReLU function is unbounded in $[0, \infty]$. Obviously, simply avoiding such unbounded functions is one option, another is to clip values outside an upper bound [209, 128] or dynamically update the clipping threshold for each layer and set the scaling factor for quantization accordingly [27].

Mixed Precision Training Mixed Precision Training (MPT) is often used to train quantized networks, whereby some values remain in full-precision so that performance is maintained and some of the aforementioned problems (e.g overflows) do not cause divergent training. It has also been observed that activations are more sensitive to quantization than weights [209]

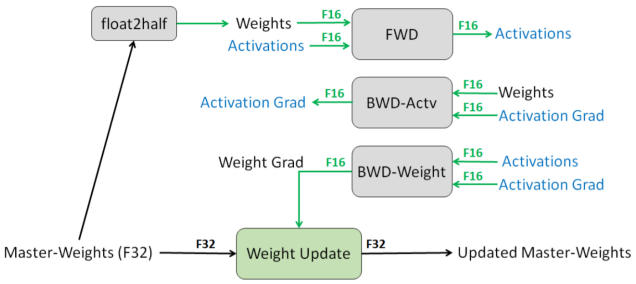


Figure 20: Mixed Precision Training (original source: Micikevicius et al. [124]

Micikevicius et al. [124] use half-precision (16-bit) floating point accuracy to represent weights, activations and gradients, without losing model accuracy or having to modify hyperparameters, almost halving the memory requirements. They round a single-precision copy of the weights for forward and backward passes after performing gradient-updates, use loss-scaling to preserve small magnitude gradient values and perform half-precision computation that accumulates into single-precision outputs before storing again as half-precision in memory.

Figure 20 illustrates MPT, where the forward and backward passes are performed with FP-16 precision copies. Once the backward pass is performed the computed FP-16 gradients are used to update the original FP-32 precision master weight. After training, the quantized weights are used for inference along with quantized activation units. This can be used in any type of layer, convolutional or fully-connected.

Others have focused solely on quantizing weights, keeping the activations at FP32 [106, 210].During gradient descent, Zhu et al. [210] learn both the quantized ternary weights and pick which of these values is assigned to each weight, represented in a codebook.

Das et al. [36] propose using Integer Fused-Multiply-and-Accumulate (FMA) operations to accumulate results of multiplied INT-16 values into INT-32 outputs and use dynamic fixed point scheme to use in tensor operations. This involves the use of a shared tensor-wide exponent and down-conversion on the maximum value of an output tensor at each given training iteration using stochastic, nearest and biased rounding. They also deal with overflow by proposing a scheme that accumulates INT-32 intermediate results to FP-32 and can trade off between precision and length of the accumulate chain to improve accuracy on the image classification tasks. They argue that previous reported results on mixed-precision integer training report on non-SoTA architectures and less difficult image tasks and hence they also report their technique on SoTA architectures for the ImageNet 1K dataset.

Quantizing by Adapting the Network Structure To further improve over mixed-precision training, there has been recent work that have aimed at better simulating the effects of quantization during training.

Mishra and Marr [127] combine low bit precision and knowledge distillation using three different schemes: (1) a low-precision (4-bit) ResNet network is trained from a full-precision ResNet network both from scratch, (2) a full precision trained network is transferred to train a low-precision network from scratch and (3) a trained full-precision network guides a smaller full-precision student randomly initialized network which is gradually becomes lower precision throughout training. They find that (2) converges faster when supervised by an already trained network and (3) outperforms (1) and set at that time was SoTA for Resnet classifiers at ternary and 4-bit precision.

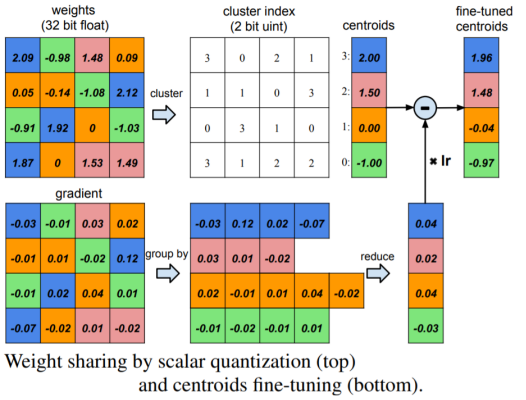
Lin et al. [114] replace FP-32 convolutions with multiple binary convolutions with various scaling factors for each convolution, overall resulting in a large range.

Zhou et al. [209] and Choi et al. [27] have both reported that the first and last convolutional layers are most sensitive to quantization and hence many works have avoided quantization on such layers. However, Choi et al. [27] find that if the quantization is not very low (e.g 8-bit integers) then these layers are expressive enough to maintain accuracy.

Zhou et al. [206] have overcome this problem by iteratively quantizing the network instead of quantize the whole model at once. During the retraining of an FP-32 model, each layer is iteratively

quantized over consecutive epochs. They also consider using supervision from a teacher network to learn a smaller quantized student network, combining knowledge distillation with quantization for further reductions.

Quantization with Pruning & Huffman Coding Coding schemes can be used to encode information in an efficient manner and construct codebooks that represent weight values and activation bit spacing. Han et al. [63] use pruning with quantization and huffman encoding for compression of ANNs by 35-49 times (9-13 times for pruning, quantization represents the weights in 5 bits instead of 32) the original size without affecting accuracy.



Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

Figure 21: original source: Han et al. [63]

Once the pruned network is established, the parameter are quantized to promote parameter sharing. This multi-stage compression strategy is illustrated in Figure 21, showing the combination of weight sharing (top) and fine-tuning of centroids (bottom). They note that too much pruning on channel level sparsity (as opposed to kernel-level) can effect the network's representational capacity.

6.4.1 Loss-aware quantization

Hou et al. [76] propose a proximal Newton algorithm with a diagonal Hessian approximation to minimize the loss with respect to the binarized weights $\hat{\boldsymbol{w}} = \alpha \boldsymbol{b}$, where $\alpha > 0$ and $\boldsymbol{b}$ is binary. During training, α is computed for the l-th layer at the t-th iteration as $\alpha_l^t = ||\boldsymbol{d}^{t-1} \otimes \boldsymbol{w}_l^t||_1 / ||\boldsymbol{d}_l^{t-1}||_1$ where $\boldsymbol{d}_l^{t-1} := \text{diag}(\mathbf{D}_l^{t-1})$ and $\boldsymbol{b}_l^t = \text{sign}(\boldsymbol{w}_l^t)$. The input is then rescaled for layer l as $\tilde{\boldsymbol{x}}_l^t = \alpha_l^t \boldsymbol{x}_{l-1}^t$ and then compute $\boldsymbol{z}_l^t$ with input $\tilde{x}_{l-1}^t$ and binary weight $\boldsymbol{b}_l^t$.

Equation 74 shows the proximal newton update step where w_l^t is the weight update at iteration t for layer l, $\mathbf{D}$ is an approximation to the diagonal of the Hessian which is already given as the 2^{nd} momentum of the adaptive momentum (adam) optimizer. The t-th iteration of the proximal Newton update is as follows:

$$\begin{aligned} \min_{\hat{\boldsymbol{w}}^t} \nabla \ell(\hat{w}^{t-1})^T (\hat{\boldsymbol{w}}^t - \hat{\boldsymbol{w}}^{t-1}) + (\hat{\boldsymbol{w}}^t - \hat{\boldsymbol{w}}^{l-1}) D^{t-1} (\hat{\boldsymbol{w}}^t - \hat{\boldsymbol{w}}^{t-1}) \\ s.t. \hat{\boldsymbol{w}}_t^l = \alpha_l^t \boldsymbol{b}_l^t, \alpha_l^t > 0,\ \boldsymbol{b}_l^t \in \{+/-1\} n_l, l = 1, \ldots, L. \end{aligned} \tag{74}$$

where the loss ℓ w.r.t binarized version of $\ell(w_t)$ is expressed in terms of the 2^{nd}-order TS expansion using a diagonal approximation of the Hessian $\mathbf{H}^{t-1}$, which estimates of the Hessian at $\boldsymbol{w}^{t-1}$. Similar to the 2^{nd} order approximations discussed in subsection 3.2, the Hessian is essential since ℓ is often flat in some directions but highly curved in others.

Explicit Loss-Aware Quantization Zhou et al. [207] propose an Explicit Loss-Aware Quantization (ELQ) method that minimizes the loss perturbation from quantization in an incremental way for very low bit precision i.e binary and ternary. Since going from FP-32 to binary or ternary bit representations can cause considerable fluctuations in weight magnitudes and in turn the predictions, ELQ directly incorporates this quantization effect in the loss function as

$$\min_{\hat{\mathbf{W}}_l} +a_1 \mathcal{L}_p(\mathbf{W}_l, \hat{\mathbf{W}}_l) + E(\mathbf{W}_l, \hat{\mathbf{W}}_l) \quad s.t. \hat{W} \in \{a_l c_k | 1 \leq k \leq K\},\ 1 \leq l \leq L \tag{75}$$

where L_p is the loss difference between quantized and the original model $||\mathcal{L}(\mathbf{W}_l) - \mathcal{L}(\hat{\mathbf{W}}_l)||$, E is the reconstruction error between the quantized and original weights $||\mathbf{W}_l - \hat{\mathbf{W}}_l||^2$, a_l a regularization coefficient for the l-th layer and c_k is an integer and k is the number of weight centroids.

Value-aware quantization Park et al. [141] like prior work mentioned in this work have also succeeded in reduced precision by reducing the dynamic range by narrowing the range where most of the weight values concentrate. Different to other work, they assign higher precision to the outliers as opposed to mapping them to the extremum of the reduced range. This small difference allow 3-bit activations to be used in ResNet-152 and DenseNet-201, leading to a 41.6% and 53.7% reduction in network size respectively.

6.4.2 Differentiable Quantization

When considering fully-differentiable training with quantized weight and activations, it is not obvious how to back-propagate through the quantization functions. These functions are discrete-valued, hence their derivative is 0 almost everywhere. So, using their gradients as-is would severely hinder the learning process. A commonly used approximation to overcome this issue is the "straight-through estimator" (STE) [71, 13], which simply passes the gradient through these functions as-is, however there has been a plethora of other techniques proposed in recent years which we describe below.

Differentiable Soft Quantization Gong et al. [53] have proposed differentiable soft quantization (DSQ) learn clipping ranges in the forward pass and approximating gradients in the backward pass. To approximate the derivative of a binary quantization function, they propose a differentiable asymptotic function (i.e smooth) which is closer to the quantization function that it is to a full-precision `tanh` function and therefore will result in less of a degradation in accuracy when converted to the binary quantization function post-training.

For multi-bit uniform quantization, given the bit width b and the floating-point activation/weight $\boldsymbol{x}$ following in the range (l, u), the complete quantization-dequantization process of uniform quantization can be defined as: $Q_U(\boldsymbol{x}) = \text{round}(\boldsymbol{x}\Delta)\Delta$ where the original range (l, u) is divided into $2^b - 1$ intervals $\mathcal{P}_i, i \in (0, 1, \dots 2^b - 1)$, and $\Delta = \frac{u-l}{2^b-1}$ is the interval length.

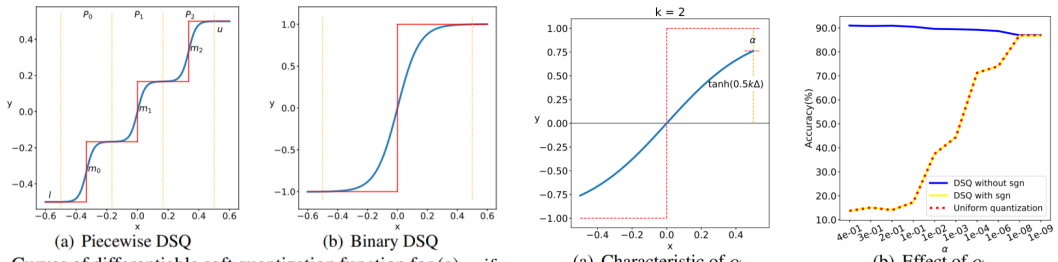
The DSQ function, shown in Equation 76, handles the point $\boldsymbol{x}$ depending what interval in $\mathcal{P}_i$ lies.

$$\phi(x) = s \tanh(k(\boldsymbol{x} - m_i)), \quad \text{if} \quad \boldsymbol{x} \in \mathcal{P}_i \tag{76}$$

with

$$m_i = l + (i + 0.5)\Delta \quad \text{and} \quad s = 1 \tanh(0.5k\Delta) \tag{77}$$

The scale parameter s for the tanh function φ ensures a smooth transitions between adjacent bit values, while k defines the functions shape where large k corresponds close to consequtive step functions given by uniform quantization with multiple piecewise levels, as shown in Figure 22a. The DSQ function then approximates the uniform quantizer φ as follows:



(a) Piecewise DSQ (b) Binary DSQ

Curves of differentiable soft quantization function for (a) uniform quantization, (b) binary quantization, a special case of (a).

(a) Characteristic of α (b) Effect of α

The characteristic variable α in DSQ.

(a) original source: Gong et al. [53]

(b) original source: Gong et al. [53]

Figure 22: Differentiable Soft Quantization

$$\mathcal{Q}_S(\boldsymbol{x}) = \begin{cases} l, & \boldsymbol{x} < l, \\ u, & \boldsymbol{x} > u, \\ l + \Delta(i + \frac{\varphi(x)+1}{2}), & \boldsymbol{x} \in \mathcal{P}_i \end{cases}$$

The DSQ can be viewed as aligning the data with the quantization values with minimal quantization error due to the bit spacing that is carried out to reflect the weight and activation distributions. Figure 22b shows the DSQ curve without [-1, 1] scaling, noting standard quantization is near perfectly approximated when the largest value on the curve bounded by +1 is small. They introduce a characteristic variable $\alpha := 1 - \tanh(0.5k\Delta) = 1 - \frac{1}{s}$ and given that,

$$\Delta = \frac{u - l}{2^b - 1} \tag{78}$$

$$\varphi(0.5\Delta) = 1 \quad \Rightarrow \quad k = \frac{1}{\Delta} \log(2/\alpha - 1) \tag{79}$$

DSQ can be used as a piecewise uniform quantizer and when only one interval is used, it is the equivalent of using DSQ for binarization.

Soft-to-hard vector quantization Agustsson et al. [3] propose to compress both the feature representations and the model by gradually transitioning from soft to hard quantization during retraining and is end-to-end differentiable. They jointly learn the quantization levels with the weights and show that vector quantization can be improve over scalar quantization.

$$H(E(\boldsymbol{Z})) = -X_{e \in [L]} m P(E(\boldsymbol{Z}) = e) \log(P(E(\boldsymbol{Z}) = e)) \tag{80}$$

They optimize the rate distortion trade-off between the expected loss and the entropy of $\mathbb{E}(Z)$:

$$\min_{E,D,\boldsymbol{W}} \mathbb{E}_{X,Y}[\ell(\hat{F}(X), Y) + \lambda R(\mathbf{W})] + \beta H(E(\boldsymbol{Z})) \tag{81}$$

Iterative Product Quantization (iPQ) Quantizing a whole network at once can be too severe for low precision (< 8 bits) and can lead to *quantization drift* - when scalar of vector quantization leads to an accumulation of reconstruction errors within the network that compound and lead to large performance degradations. To combat this, Stock et al. [170] iteratively quantize the network starting with low layers and only performing gradient updates on the rest of the remaining layers until they are robust to the quantized layers. This is repeated until quantization is carried out on the last layer, resulting in the whole network being amenable to quantization.

The codebook is updated by averaging the gradients of the weights within the block b_{KL} as

$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \eta \frac{1}{|J_c|} \sum_{(k,l) \in J_c} \frac{\partial \mathcal{L}}{\partial b_{\text{KL}}} \quad \text{where} \quad J_{\boldsymbol{c}} = \{(k,l) \mid c[\mathbf{I}_{\text{KL}}] = \boldsymbol{c}\} \tag{82}$$

where $\mathcal{L}$ is the loss function, I_{KL} is an index for the (k, l) subvector and $\eta > 0$ is the codebook learning rate. This adapts the upper layers to the drift appearing in their inputs, reducing the impact of the quantization approximation on the overall performance.

Quantization-Aware Training Instead of iPQ, Jacob et al. [84] use a straight through estimator [(STE) 13] to backpropogate through quantized weights and activations of convolutional layers during training. Figure 23 shows the 8-bit weights and activations, while the accumulator is represented in 32-bit integer.

They also note that in order to have a challenging architecture to compress, experiments should move towards trying to compress architectures which are already have a minimal number of parameter and perform relatively well to much larger predecessing architectures e.g EfficientNet, SqueezeNet and ShuffleNet.

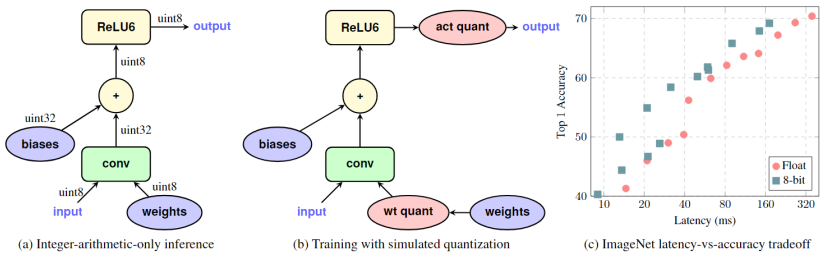


Figure 23: original source ([84]): Integer-arithmetic-only quantization

Quantization Noise Fan et al. [47] argue that both iPQ and QAT are less suitable for very low precision such as INT4, ternary and binary. They instead propose to randomly simulate quantization noise on a subset of the network and only perform backward passes on the remaining weights in the network. Essentially this is a combination of DropConnect (instead of the Bernoulli function, it is a quantization noise function) and Straight Through Estimation is used to backpropogate through the sample of subvectors chosen for quantization for a given mini-batch.

Estimating quantization noise through randomly sampling blocks of weights to be quantized allows the model to become robust to very low precision quantization without being too severe, as is the case with previous quantization-aware training [84]. The authors show that this iterative quantization approach allows large compression rates in comparison to QAT while staying close to (few perplexity points in the case of language modelling and accuracy for image classification) the uncompressed model in terms of performance. They reach SoTA compression and accuracy tradeoffs for language modelling (compression of Transformers such as RoBERTa on WikiText) and image classification (compressing EfficientNet-B3 by 80% on ImageNet).

Hessian-Based Quantization The precision and order (by layer) of quantization has been chosen using 2^{nd} order information from the Hessian [43]. They show that on already relatively small CNNs (ResNet20, Inception-V3, SqueezeNext) that Hessian Aware Quantization (HAWQ) training leads to SoTA compression on CIFAR-10 and ImageNet with a compression ratio of 8 and in some cases exceed the accuracy of the original network with no quantization.

Similarly, Shen et al. [164] quantize transformer-based models such as BERT with mixed precision by also using 2^{nd} order information from the Hessian matrix. They show that each layer exhibits varying amount of information and use a sensitivity measure based on mean and variance of the top eigenvalues. They show the loss landscape as the two most dominant eigenvectors of the Hessian are perturbed and suggest that layers that shower a smoother curvature can undergo lower but precision. In the cases of MNLI and CoNLL datasets, upper layers closer to the output show flatter curvature in comparison to lower layers. From this observation, they are motivated to perform a group-wise quantization scheme whereby blocks of a matrix have different amounts of quantization with unique quantization ranges and look up table. A Hessian-based mixed precision scheme is then used to decide which blocks of each matrix are assigned the corresponding low-bit precisions of varying ranges and analyse the differences found for quantizing different parts of the self-attention block (self-attention matrices and fully-connected feedforward layers) and their inputs (embeddings) and find the highest compression ratios can be attributed to most of the parametes in the self-attention blocks.

7 Summary

The above sections have provided descriptions of old and new compression methods and techniques. We finish by providing general recommendations for this field and future research directions that I deem to be important in the coming years.

7.1 Recommendations

Old Baselines May Still Be Competitive Evidently, there has been an extensive amount of work in pruning, quantization, knowledge distillation and combinations of the aforementioned for neural

networks. We note that many of these approaches, particularly pruning, were proposed in decades past [29, 131, 65, 103, 191, 90, 151, 46]. The current trend of deep neural networks growing ever larger means that keeping track of new innovations on the topic of reducing network size becomes increasingly important. Therefore, we suggest that comparing past and present techniques for compression should be standardized across models, datasets and evaluation metrics such that these comparisons are made direct. Ideally this would be carried out using the same libraries in the same language (e.g PyTorch or Tensorflow in Python) to further minimize any implementation differences that naturally occur.

More Compression Work on Large Non-Sparse Architectures The majority of the aforementioned compression techniques that have been proposed are the context of CNNs since they have been used extensively over the past 3 decades, predominantly for image-based tasks. We suggest that future and existing techniques can now also be extended to recent architectures such as Transformers and applied to other important tasks (e.g text generation, speech recognition). In fact, this already becoming apparent by the rise in the number of papers around compressing transformers in the NLP community. More specifically, reducing the size BERT and related models, as discussed in subsection 5.3.

Challenging Compression on Already Parameter Efficient Architectures The importance of trying to compress already parameter efficient architectures (e.g EffecientNet, SqueezNet or MobileNet for CNNs or DistilBERT for Transformers), such as those discussed in Appendix A, makes for more challenging compression problem. Although compressing large overparameterized network have a large and obvious capacity for compression, compressing already parameter efficient network provides more insight into the advantages and disadvantages of different compression techniques.

7.2 Future Research Directions

The field of neural network compression has seen a resurgence in activity given the growing size of state of the art of models that are pushing the boundaries of hardware and practitioners resources. However, compression techniques are still in a relatively early stage of development. Below, I discuss a few research directions I think are worth exploring for the future of model compression.

What Combination of Compression Techniques To Use ? Most of the works discussed here have not used multiple compression techniques for retraining (e.g pruning with distillation and quantization) nor have they figured out what order is optimal for a given set of tasks and architectures. Han et al. [63] is a prime example of combining compression techniques, combining quantization, pruning and huffman coding. However, it still remains unclear what combination and what order should be used to get the desired compression tradeoff between performance, speed and storage. A strong ablation study on many different architectures with various combinations and orders would be greatly insightful from a practical standpoint.

Automatically Choosing Student Size in Knowledge Distillation Current knowledge distillation approaches use fixed sized students during retraining. However, to get the desired tradeoff between performance versus student network size it requires a manual iteration over different student size in retraining. This is often used to visualize this tradeoff in papers, however automatically searching for student architecture during knowledge distillation is certainly an area of future research worth considering. In this context, meta learning and neural architecture search becomes important topics to bridge this gap between manually found student architectures to automatic techniques for finding the architectures.

Few-Shot Knowledge Distillation In cases where larger pretrained models are required for a set (or single) of target tasks where there are only few samples, knowledge distillation can be used to distill the knowledge of teacher specifically for that transfer domain. The advantage of doing so is that we benefit from the transferability of teacher network while also distilling these large feature sets into a smaller network.

Meta Learning Based Compression Meta learning [162, 4] have been successfully used for *learning to learn*. Meta-learning how these larger teacher networks learn could be beneficial for

improving the performance and convergence of a distilled student network. To date, I believe this is an unexplored area of research.

Further Theoretical Analysis Recent work has aided in our understanding of generalization in deep neural networks [135, 187, 133, 11, 39, 160] and proposed measures for tracking generalization performance while training DNNs. Further theoretical analysis of compression generalization is a worthwhile endeavour considering the growing importance and usage of compressing already trained neural networks. This is distinctly different than training models from random initialization and requires a new generalization paradigm to understand how compression works for each type (i.e pruning, quantization etc.).

References

[1] Prem Raj Adhikari and Jaakko Hollmen. 2012. Multiresolution mixture modeling using merging of mixture components. In *Asian Conference on Machine Learning*. pages 17–32.

[2] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. 2019. Compressing gans using knowledge distillation. *arXiv preprint arXiv:1902.00159* .

[3] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. 2017. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems*. pages 1141–1151.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*. pages 3981–3989.

[5] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13(3):32.

[6] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. 2017. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030* .

[7] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Advances in neural information processing systems*. pages 2654–2662.

[8] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682* .

[9] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. In *Advances in Neural Information Processing Systems*. pages 688–699.

[10] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. 2018. Aciq: Analytical clipping for integer quantization of neural networks. *openreview.net* .

[11] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 116(32):15849–15854.

[12] Mikhail Belkin, Daniel Hsu, and Ji Xu. 2019. Two models of double descent for weak features. *arXiv preprint arXiv:1903.07571* .

[13] Yoshua Bengio, Nicholas Leonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* .

[14] 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. 2020. Language models are few-shot learners.

[15] Charles G Broyden. 1965. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation* 19(92):577–593.

[16] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pages 535–541.

[17] Andres Buzo, A Gray, R Gray, and John Markel. 1980. Speech coding based upon vector quantization. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28(5):562–574.

[18] Leopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H Elibol, and Hanlin Tang. 2020. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. *arXiv preprint arXiv:2001.05674* .

[19] Erick Cantu-Paz. 2003. Pruning neural networks with distribution estimation algorithms. In *Genetic and Evolutionary Computation Conference*. Springer, pages 790–800.

[20] Carlos M Carvalho, Nicholas G Polson, and James G Scott. 2010. The horseshoe estimator for sparse signals. *Biometrika* 97(2):465–480.

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

[22] Sneha Chaudhari, Gungor Polatkan, Rohan Ramanath, and Varun Mithal. 2019. An attentive survey of attention models. *arXiv preprint arXiv:1904.02874* .

[23] Yevgen Chebotar and Austin Waters. 2016. Distilling knowledge from ensembles of neural networks for speech recognition. In *Interspeech*. pages 3439–3443.

[24] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International conference on machine learning*. pages 2285–2294.

[25] Yen-Chun Chen, Zhe Gan, Yu Cheng, Jingzhou Liu, and Jingjing Liu. 2019. Distilling the knowledge of bert for text generation. *arXiv preprint arXiv:1911.03829* .

[26] Jang Hyun Cho and Bharath Hariharan. 2019. On the efficacy of knowledge distillation. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 4794–4802.

[27] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* .

[28] Francois Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 1251–1258.

[29] John Cleary and Ian Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE transactions on Communications* 32(4):396–402.

[30] Yaim Cooper. 2018. The loss landscape of overparameterized neural networks. *arXiv preprint arXiv:1804.10200* .

[31] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* .

[32] Jia Cui, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, Tom Sercu, Kartik Audhkhasi, Abhinav Sethy, Markus Nussbaum-Thom, and Andrew Rosenberg. 2017. Knowledge distillation across ensembles of multilingual models for low-resource languages. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pages 4825–4829.

[33] Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*. volume 33, pages 6292–6299.

[34] Bin Dai, Chen Zhu, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. *arXiv preprint arXiv:1802.10399* .

[35] William Dally. 2015. High-performance hardware for machine learning. *NIPS Tutorial* .

[36] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, et al. 2018. Mixed precision training of convolutional neural networks using integer operations. *arXiv preprint arXiv:1802.00930* .

[37] Lieven De Lathauwer. 2008. Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications* 30(3):1033–1066.

[38] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819* .

[39] Michal Derezinski, Feynman Liang, and Michael W Mahoney. 2019. Exact expressions for double descent and implicit regularization via surrogate random design. *arXiv preprint arXiv:1912.04533* .

[40] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* .

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

[42] Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*. pages 4857–4867.

[43] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 293–302.

[44] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. 2018. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054* .

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

[46] Scott E Fahlman and Christian Lebiere. 1990. The cascade-correlation learning architecture. In *Advances in neural information processing systems*. pages 524–532.

[47] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression.

[48] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* .

[49] Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation. *arXiv preprint arXiv:1702.01802* .

[50] Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* 36(4):193–202.

[51] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pages 1243–1252.

[52] David E Goldberg and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, Elsevier, volume 1, pages 69–93.

[53] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 4852–4861.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[55] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. 2013. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082* .

[56] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. 2019. Nat: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*. pages 735–747.

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

[58] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems* 29(11):5784–5789.

[59] Masafumi Hagiwara. 1993. Removal of hidden units and weights for back propagation networks. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. IEEE, volume 1, pages 351–354.

[60] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53(2):217–288.

[61] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*. pages 8527–8537.

[62] Hong-Gui Han and Jun-Fei Qiao. 2013. A structure optimisation algorithm for feedforward neural network construction. *Neurocomputing* 99:347–357.

[63] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* .

[64] Babak Hassibi and David G Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*. pages 164–171.

[65] Babak Hassibi, David G Stork, and Gregory Wolff. 1994. Optimal brain surgeon: Extensions and performance comparisons. In *Advances in neural information processing systems*. pages 263–270.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 770–778.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, pages 630–645.

[68] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*. pages 784–800.

[69] Srinidhi Hegde, Ranjitha Prasad, Ramya Hebbalaguppe, and Vishwajith Kumar. 2019. Variational student: Learning compact and sparser networks in knowledge distillation framework. *arXiv preprint arXiv:1910.12061* .

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

[71] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning. *Coursera, video lectures* 264:1.

[72] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* .

[73] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* .

[74] Frank L Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics* 6(1-4):164–189.

[75] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

[76] Lu Hou, Quanming Yao, and James T Kwok. 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600* .

[77] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* .

[78] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*. pages 2042–2050.

[79] Dichao Hu. 2019. An introductory survey on attention mechanisms in nlp problems. In *Proceedings of SAI Intelligent Systems Conference*. Springer, pages 432–448.

[80] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. 2018. A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394* .

[81] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 4700–4708.

[82] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360* .

[83] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* .

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

[85] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* .

[86] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33(1):117–128.

[87] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* .

[88] Thomas Kailath. 1980. *Linear systems*, volume 156. Prentice-Hall Englewood Cliffs, NJ.

[89] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. 2019. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322* .

[90] Ehud D Karnin. 1990. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks* 1(2):239–242.

[91] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*. IEEE, volume 4, pages 1942–1948.

[92] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. 2019. A survey of the recent architectures of deep convolutional neural networks. *arXiv preprint arXiv:1901.06032* .

[93] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* .

[94] Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947* .

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

[96] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* .

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. pages 1097–1105.

[98] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683* .

[99] Samuli Laine and Timo Aila. 2016. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242* .

[100] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* .

[101] Philippe Lauret, Eric Fock, and Thierry Alex Mara. 2006. A node pruning algorithm based on a fourier amplitude sensitivity test method. *IEEE transactions on neural networks* 17(2):273–293.

[102] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 2554–2564.

[103] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. pages 598–605.

[104] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* .

[105] Asriel U Levin, Todd K Leen, and John E Moody. 1994. Fast pruning using principal components. In *Advances in neural information processing systems*. pages 35–42.

[106] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* .

[107] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* .

[108] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*. pages 6389–6399.

[109] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. 2017. Learning from noisy labels with distillation. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 1910–1918.

[110] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*. pages 2849–2858.

[111] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. In *Advances in Neural Information Processing Systems*. pages 2181–2191.

[112] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2018. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*. pages 2425–2432.

[113] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. 2019. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 2790–2799.

[114] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*. pages 345–353.

[115] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* .

[116] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arXiv preprint arXiv:1904.09482* .

[117] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* .

[118] Stuart Lloyd. 1982. Least squares quantization in pcm. *IEEE transactions on information theory* 28(2):129–137.

[119] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. 2017. Data-free knowledge distillation for deep neural networks. *arXiv preprint arXiv:1710.07535* .

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

[121] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Advances in neural information processing systems*. pages 7816–7827.

[122] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. 2019. Mixed precision training with 8-bit floating point. *arXiv preprint arXiv:1905.12334* .

[123] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. 2016. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981* .

[124] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740* .

[125] Szymon Migacz. 2017. 8-bit inference with tensorrt. In *GPU technology conference*. volume 2, page 5.

[126] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. 2019. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393* .

[127] Asit Mishra and Debbie Marr. 2017. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852* .

[128] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. 2017. Wrpn: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134* .

[129] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369* .

[130] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440* 3.

[131] Michael C Mozer and Paul Smolensky. 1989. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*. pages 107–115.

[132] Rafael Muller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems*. pages 4696–4705.

[133] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2019. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292* .

[134] Pramod L Narasimha, Walter H Delashmit, Michael T Manry, Jiang Li, and Francisco Maldonado. 2008. An integrated growing-pruning method for feedforward network training. *Neurocomputing* 71(13-15):2831–2847.

[135] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. 2018. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076* .

[136] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. In *Advances in neural information processing systems*. pages 442–450.

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

[138] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor. 2019. Asap: Architecture search, anneal and prune. *arXiv preprint arXiv:1904.04123* .

[139] Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing* 33(5):2295–2317.

[140] Daniel S. Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V. Le. 2020. Improved noisy student training for automatic speech recognition.

[141] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. 2018. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. pages 580–595.

[142] Sungrae Park, JunKeon Park, Su-Jin Shin, and Il-Chul Moon. 2018. Adversarial dropout for supervised and semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[143] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. 2019. Relational knowledge distillation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 3967–3976.

[144] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. *arXiv preprint arXiv:1802.05751* .

[145] Mary Phuong and Christoph Lampert. 2019. Towards understanding knowledge distillation. In *International Conference on Machine Learning*. pages 5142–5151.

[146] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* .

[147] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* .

[148] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2019. Zero: Memory optimization towards training a trillion parameter models. *arXiv preprint arXiv:1910.02054* .

[149] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* .

[150] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, pages 525–542.

[151] Russell Reed. 1993. Pruning algorithms-a survey. *IEEE transactions on Neural Networks* 4(5):740–747.

[152] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. 2013. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 2754–2761.

[153] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* .

[154] C Rosset. 2019. Turing-nlg: A 17-billion-parameter language model by microsoft. *Microsoft Blog* .

[155] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[156] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, pages 6655–6659.

[157] Victor Sanh. 2019. Smaller, faster, cheaper, lighter: Introducing distilbert, a distilled version of bert. https://medium.com/huggingface/distilbert-8cf3380435b5.

[158] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* .

[159] Bharat Bhusan Sau and Vineeth N Balasubramanian. 2016. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650* .

[160] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. 2019. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment* 2019(12):124020.

[161] Benjamin Scellier and Yoshua Bengio. 2017. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience* 11:24.

[162] Jurgen Schmidhuber. 1987. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Ph.D. thesis, Technische Universitat Munchen.

[163] Rudy Setiono and Wee Kheng Leow. 2000. Pruned neural networks for regression. In *Pacific Rim International Conference on Artificial Intelligence*. Springer, pages 500–509.

[164] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Q-bert: Hessian based ultra low precision quantization of bert. *arXiv preprint arXiv:1909.05840* .

[165] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. 2009. Hash kernels for structured data. *Journal of Machine Learning Research* 10(Nov):2615–2637.

[166] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053* .

[167] Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810* .

[168] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *Journal of Machine Learning Research*.

[169] Xuemeng Song, Fuli Feng, Xianjing Han, Xin Yang, Wei Liu, and Liqiang Nie. 2018. Neural compatibility modeling with attentive knowledge distillation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. pages 5–14.

[170] Pierre Stock, Armand Joulin, Remi Gribonval, Benjamin Graham, and Herve Jegou. 2019. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686* .

[171] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355* .

[172] Mingxing Tan and Quoc V. Le. 2019. Efficientnet: Improving accuracy and efficiency through automl and model scaling. https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html.

[173] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* .

[174] Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*. pages 1195–1204.

[175] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszar. 2018. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787* .

[176] Yonglong Tian, Dilip Krishnan, and Phillip Isola. 2019. Contrastive representation distillation. *arXiv preprint arXiv:1910.10699* .

[177] Juanjuan Tu, Yongzhao Zhan, and Fei Han. 2010. A neural network pruning method optimized with pso algorithm. In *2010 Second International Conference on Computer Modeling and Simulation*. IEEE, volume 3, pages 257–259.

[178] Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31(3):279–311.

[179] Frederick Tung and Greg Mori. 2019. Similarity-preserving knowledge distillation. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 1365–1374.

[180] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008* .

[181] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. pages 5998–6008.

[182] Christopher A Walsh. 2013. Peter huttenlocher (1931–2013).

[183] Weishui Wan, Shingo Mabu, Kaoru Shimada, Kotaro Hirasawa, and Jinglu Hu. 2009. Enhancing the generalization ability of neural networks through controlling the hidden layers. *Applied Soft Computing* 9(1):404–414.

[184] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* .

[185] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*. pages 7675–7684.

[186] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. 2018. Kdgan: Knowledge distillation with generative adversarial networks. In *Advances in Neural Information Processing Systems*. pages 775–786.

[187] Colin Wei, Jason Lee, Qiang Liu, and Tengyu Ma. 2018. On the margin theory of feedforward neural networks. *OpenReview* .

[188] Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. 1991. Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems*. pages 875–882.

[189] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. pages 1113–1120.

[190] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. pages 2074–2082.

[191] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. 1990. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing* 14(3):347–361.

[192] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024* .

[193] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 1492–1500.

[194] Jian Xue, Jinyu Li, and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*. pages 2365–2369.

[195] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. 2014. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. pages 6359–6363.

[196] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*. pages 5754–5764.

[197] Amir Yazdanbakhsh, Ahmed T Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, and Hadi Esmaeilzadeh. 2018. Releq: An automatic reinforcement learning approach for deep quantization of neural networks. *arXiv preprint arXiv:1811.01704* .

[198] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. 2018. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 9378–9387.

[199] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. 2017. Long-term forecasting using tensor-train rnns. *Arxiv* .

[200] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 9194–9203.

[201] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 7370–7379.

[202] Sergey Zagoruyko and Nikos Komodakis. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* .

[203] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* .

[204] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. 2018. Learning to share: Simultaneous parameter tying and sparsification in deep learning. *OpenReview.net* .

[205] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 6848–6856.

[206] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* .

[207] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. 2018. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 9426–9435.

[208] Chunting Zhou, Graham Neubig, and Jiatao Gu. 2019. Understanding knowledge distillation in non-autoregressive machine translation. *arXiv preprint arXiv:1911.02727* .

[209] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* .

[210] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* .

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

[212] Xiatian Zhu, Shaogang Gong, et al. 2018. Knowledge distillation by on-the-fly native ensemble. In *Advances in neural information processing systems*. pages 7517–7527.

Appendices

A Low Resource and Efficient CNN Architectures

A.0.1 MobileNet

Howard et al. [77] propose compression of convolutional neural networks for embedded and mobile vision applications using depth-wise separable convolutions (DSC) and use two hyperparameters that tradeoff latency and accuracy. DSCs factorize a standard convolution into a depthwise convolution and 1×1 pointwise convolution. Each input channel is passed through a DSC filter followed by a pointwise 1×1 convolution that combines the outputs of the DSC. Unlike standard convolutions, DSCs split the convolution into two steps, first filtering then combining outputs of each DSC filter, which is why this is referred to as a factorization approach.

Experiments on ImageNet image classification demonstrated that these smaller networks can achieve ac curacies similar to much larger networks.

A.0.2 SqueezeNet

Iandola et al. [82] reduce the network architecture by reducing 3×3 filters to 1×1 filters (squeeze layer), reduce the number of input channels to 3×3 filters using squeeze layers and downsample later in the network to avoid the bottleneck of information through the network too early and in turn lead to better performance. A *fire* module is made up of the squeeze layer is into an *expand* layer that is a mix of 1×1 and 3×3 convolution filters and the number of filters per *fire* module is increased as it gets closer to the last layer.

By using these architectural design decisions, SqueezeNet can compete with AlexNet with 50 times smaller network and even outperforms layer decomposition and pruning for deep compression. When combined with INT8 quantization, SqueezeNet yields a 0.66 MB model which is 363 times smaller than 32-bit AlexNet, while still maintaining performance.

A.0.3 ShuffleNet

ShuffleNet [205] uses pointwise group convolutions [97](i.e using a different set of convolution filter groups on the same input features, this allows for model parallelization) and channel shuffles (randomly shuffling helps information flow across feature channels) to reduce compute while maintaining accuracy. ShuffleNet is made up economical 3×3 depthwise convolutional filters and replace 1×1 layer with pointwise group convolutional followed by the channel shuffle. Unlike predecessor models [193, 28], ShuffleNet is efficient for smaller networks as they find big improvements when tested on ImageNet and MSCOCO object detection using 40 Mega FLOPs and achieves 13 times faster training over AlexNet without sacrificing much accuracy.

A.0.4 DenseNet

Gradients can vanish in very deep networks because the error becomes more difficult to backpropogate as the number of matrix multiplications increase. DenseNets [81] address gradient vanishing connecting the feature maps of the previous layer to the inputs of the next layer, similar to ResNet skip connections. This reusing of features mean the network efficient with its use of parameters. Although, deep and thin DenseNetworks can be parameter efficient, they do tradeoff with memory/speed efficiency in comparison to shallower yet wider network ([203]) because all layer outputs need to be stored to perform backpropogation. However, DenseNets too can be made wider and shallower to become more memory effecient if required.

B Low Resource and Efficient Transformer Architectures

In this section we describe some work that tries to find efficient architectures during training and hence are not considered compressed networks in the traditional definition as they are not already pretrained before the network is reduced.

Transformer Architecture Search Most neural architecture search (NAS) methods learn to apply modules in the network with no regard for the computational cost of adding them, such as Neural architecture optimization [121] which uses an encoder-decoder model to reconstruct an architecture from a continuous space. Guo et al. [56] instead have proposed to learn a transformer architecture while minimizing the computational burden, avoiding modules with large number of parameters if necessary. However, solving such problem is NP-hard. Therefore, they propose to treat the optimization problem as a Markov Decision Process (MDP) and optimize the policies w.r.t. to the different architectures using reinforcement learning. These different architectures are replace redundant transformations with more efficient ones such as skip connections or removing connections altogether.