Neurocomputing 461 (2021) 370–403

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Pruning and quantization for deep neural network acceleration: A survey

Tailin Liang [a,b], John Glossner [a,b,c], Lei Wang [a], Shaobo Shi [a,b], Xiaotong Zhang [a,*]

[a] School of Computer and Communication Engineering, University of Science and Technology Beijing Beijing 100083, China
[b] Hua Xia General Processor Technologies, Beijing 100080, China
[c] General Processor Technologies, Tarrytown, NY 10591, United States

ARTICLE INFO

Article history:
Received 11 March 2021
Revised 15 May 2021
Accepted 15 July 2021
Available online 21 July 2021
Communicated by Zidong Wang

Keywords:
Convolutional neural network
Neural network acceleration
Neural network quantization
Neural network pruning
Low-bit mathematics

ABSTRACT

Deep neural networks have been applied in many applications exhibiting extraordinary abilities in the field of computer vision. However, complex network architectures challenge efficient real-time deployment and require significant computation resources and energy costs. These challenges can be overcome through optimizations such as network compression. Network compression can often be realized with little loss of accuracy. In some cases accuracy may even improve. This paper provides a survey on two types of network compression: pruning and quantization. Pruning can be categorized as static if it is performed offline or dynamic if it is performed at run-time. We compare pruning techniques and describe criteria used to remove redundant computations. We discuss trade-offs in element-wise, channel-wise, shape-wise, filter-wise, layer-wise and even network-wise pruning. Quantization reduces computations by reducing the precision of the datatype. Weights, biases, and activations may be quantized typically to 8-bit integers although lower bit width implementations are also discussed including binary neural networks. Both pruning and quantization can be used independently or combined. We compare current techniques, analyze their strengths and weaknesses, present compressed network accuracy results on a number of frameworks, and provide practical guidance for compressing networks.

1. Introduction

Deep Neural Networks (DNNs) have shown extraordinary abilities in complicated applications such as image classification, object detection, voice synthesis, and semantic segmentation [138]. Recent neural network designs with billions of parameters have demonstrated human-level capabilities but at the cost of significant computational complexity. DNNs with many parameters are also time-consuming to train [26]. These large networks are also difficult to deploy in embedded environments. Bandwidth becomes a limiting factor when moving weights and data between Compute Units (CUs) and memory. Over-parameterization is the property of a neural network where redundant neurons do not improve the accuracy of results. This redundancy can often be removed with little or no accuracy loss [225].

Fig. 1 shows three design considerations that may contribute to over-parameterization: 1) network structure, 2) network optimization, and 3) hardware accelerator design. These design considerations are specific to Convolutional Neural Networks (CNNs) but also generally relevant to DNNs.

Network structure encompasses three parts: 1) novel components, 2) network architecture search, and 3) knowledge distillation. Novel components is the design of efficient blocks such as separable convolution, inception blocks, and residual blocks. They are discussed in Section 2.4. Network components also encompasses the types of connections within layers. Fully connected deep neural networks require N^2 connections between neurons. Feed forward layers reduce connections by considering only connections in the forward path. This reduces the number of connections to N. Other types of components such as dropout layers can reduce the number of connections even further.

Network Architecture Search (NAS) [63], also known as network auto search, programmatically searches for a highly efficient network structure from a large predefined search space. An estimator is applied to each produced architecture. While time-consuming to compute, the final architecture often outperforms manually designed networks.

Knowledge Distillation (KD) [80,206] evolved from knowledge transfer [27]. The goal is to generate a simpler compressed model that functions as well as a larger model. KD trains a student network that tries to imitate a teacher network. The student network

* Corresponding author.
E-mail addresses: tailin.liang@xs.ustb.edu.cn (T. Liang), jglossner@ustb.edu.cn (J. Glossner), wanglei@ustb.edu.cn (L. Wang), sbshi@hxgpt.com (S. Shi), zxt@ies.ustb.edu.cn (X. Zhang).

https://doi.org/10.1016/j.neucom.2021.07.045
0925-2312/

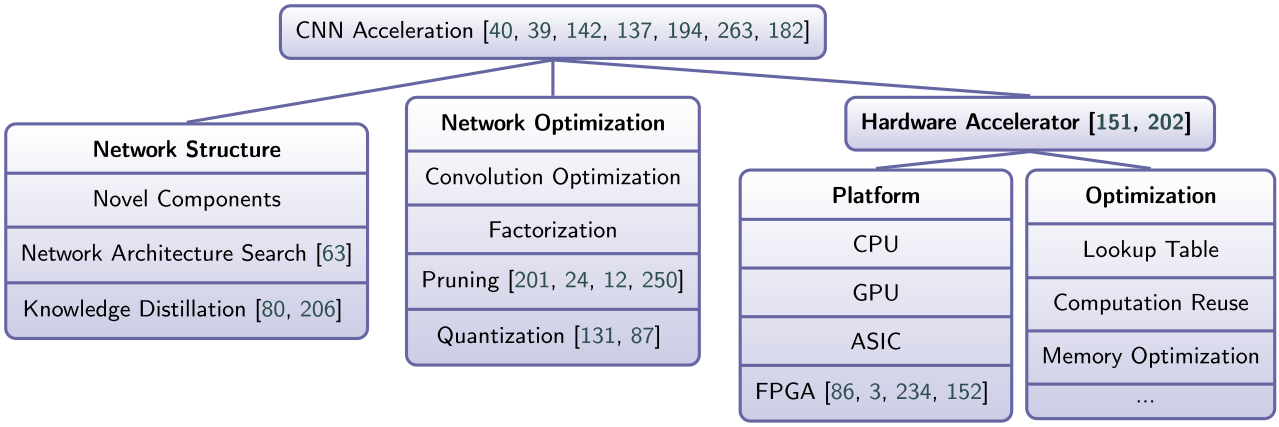


Fig. 1. CNN Acceleration Approaches: Follow the sense from designing to implementing, CNN acceleration could fall into three categories, structure design (or generation), further optimization, and specialized hardware. (see above mentioned reference for further information.)

is usually but not always smaller and shallower than the teacher. The trained student model should be less computationally complex than the teacher.

Network optimization [137] includes: 1) computational convolution optimization, 2) parameter factorization, 3) network pruning, and 4) network quantization. Convolution operations are more efficient than fully connected computations because they keep high dimensional information as a 3D tensor rather than flattening the tensors into vectors that removes the original spatial information. This feature helps CNNs to fit the underlying structure of image data in particular. Convolution layers also require significantly less coefficients compared to Fully Connected Layers (FCLs). Computational convolution optimizations include Fast Fourier Transform (FFT) based convolution [168], Winograd convolution [135], and the popular image to column (im2col) [34] approach. We discuss im2col in detail in Section 2.3 since it is directly related to general pruning techniques.

Parameter factorization is a technique that decomposes higher-rank tensors into lower-rank tensors simplifying memory access and compressing model size. It works by breaking large layers into many smaller ones, thereby reducing the number of computations. It can be applied to both convolutional and fully connected layers. This technique can also be applied with pruning and quantization.

Network pruning [201,24,12,250] involves removing parameters that don't impact network accuracy. Pruning can be performed in many ways and is described extensively in Section 3.

Network quantization [131,87] involves replacing datatypes with reduced width datatypes. For example, replacing 32-bit Floating Point (FP32) with 8-bit Integers (INT8). The values can often be encoded to preserve more information than simple conversion. Quantization is described extensively in Section 4.

Hardware accelerators [151,202] are designed primarily for network acceleration. At a high level they encompass entire processor platforms and often include hardware optimized for neural networks. Processor platforms include specialized Central Processing Unit (CPU) instructions, Graphics Processing Units (GPUs), Application Specific Integrated Circuits (ASICs), and Field Programmable Gate Arrays (FPGAs).

CPUs have been optimized with specialized Artificial Intelligence (AI) instructions usually within specialized Single Instruction Multiple Data (SIMD) units [49,11]. While CPUs can be used for training, they have primarily been used for inference in systems that do not have specialized inference accelerators.

GPUs have been used for both training and inference. nVidia has specialized tensor units incorporated into their GPUs that are optimized for neural network acceleration [186]. AMD [7], ARM [10], and Imagination [117] also have GPUs with instructions for neural network acceleration.

Specialized ASICs have also been designed for neural network acceleration. They typically target inference at the edge, in security cameras, or on mobile devices. Examples include: General Processor Technologies (GPT) [179], ARM, nVidia, and 60 + others [202] all have processors targeting this space. ASICs may also target both training and inference in datacenters. Tensor processing units (TPU) from Google [125], Habana from Intel [169], Kunlun from Baidu [191], Hanguang from Alibaba [124], and Intelligence Processing Unit (IPU) from Graphcore [121].

Programmable reconfigurable FPGAs have been used for neural network acceleration [86,3,234,152]. FPGAs are widely used by researchers due to long ASIC design cycles. Neural network libraries are available from Xilinx [128] and Intel [69]. Specific neural network accelerators are also being integrated into FPGA fabrics [248,4,203]. Because FPGAs operate at the gate level, they are often used in low-bit width and binary neural networks [178,267,197].

Neural network specific optimizations are typically incorporated into custom ASIC hardware. Lookup tables can be used to accelerate trigonometric activation functions [46] or directly generate results for low bit-width arithmetic [65], partial products can be stored in special registers and reused [38], and memory access ordering with specialized addressing hardware can all reduce the number of cycles to compute a neural network output [126]. Hardware accelerators are not the primary focus of this paper. However, we do note hardware implementations that incorporate specific acceleration techniques. Further background information on efficient processing and hardware implementations of DNNs can be found in [225].

We summarize our main contributions as follows:

- We provide a review of two network compression techniques: pruning and quantization. We discuss methods of compression, mathematical formulations, and compare current State-Of-The-Art (SOTA) compression methods.
- We classify pruning techniques into static and dynamic methods, depending if they are done offline or at runtime, respectively.
- We analyze and quantitatively compare quantization techniques and frameworks.
- We provide practical guidance on quantization and pruning.

This paper focuses primarily on network optimization for convolutional neural networks. It is organized as follows: In Section 2 we give an introduction to neural networks and specifically convolu-

tional neural networks. We also describe some of the network optimizations of convolutions. In Section 3 we describe both static and dynamic pruning techniques. In Section 4 we discuss quantization and its effect on accuracy. We also compare quantization libraries and frameworks. We then present quantized accuracy results for a number of common networks. We present conclusions and provide guidance on appropriate application use in Section 5. Finally, we present concluding comments in Section 7.

2. Convolutional neural network

Convolutional neural networks are a class of feed-forward DNNs that use convolution operations to extract features from a data source. CNNs have been most successfully applied to visual-related tasks however they have found use in natural language processing [95], speech recognition [2], recommendation systems [214], malware detection [223], and industrial sensors time series prediction [261]. To provide a better understanding of optimization techniques, in this section, we introduce the two phases of CNN deployment - training and inference, discuss types of convolution operations, describe Batch Normalization (BN) as an acceleration technique for training, describe pooling as a technique to reduce complexity, and describe the exponential growth in parameters deployed in modern network structures.

2.1. Definitions

This section summarizes terms and definitions used to describe neural networks as well as acronyms collected in Table 1.

- Coefficient - A constant by which an algebraic term is multiplied. Typically, a coefficient is multiplied by the data in a CNN filter.
- Parameter - All the factors of a layer, including coefficients and biases.
- Hyperparameter - A predefined parameter before network training, or fine-tunning (re-training).
- Activation ($\mathbf{A} \in \mathbb{R}^{h\times w\times c}$) - The activated (e.g., ReLu, Leaky, Tanh, etc.) output of one layer in a multi-layer network architecture, typically in height h, width w, and channel c. The $h \times w$ matrix is sometimes called an activation map. We also denote activation as output ($\mathbf{O}$) when the activation function does not matter.
- Feature ($\mathbf{F} \in \mathbb{R}^{h\times w\times c}$) - The input data of one layer, to distinguish the output $\mathbf{A}$. Generally the feature for the current layer is the activation of the previous layer.
- Kernel ($\mathbf{k} \in \mathbb{R}^{k_1\times k_2}$) - Convolutional coefficients for a channel, excluding biases. Typically they are square (i.e., $k_1 = k_2$) and sized 1, 3, 7.
- Filter ($\mathbf{w} \in \mathbb{R}^{k_1\times k_2\times c\times n}$) - Comprises all of the kernels corresponding to the c channels of input features. The filter's number, n, results in different output channels.
- Weights - Two common uses: 1) kernel coefficients when describing part of a network, and 2) all the trained parameters in a neural network model when discussing the entire network.

2.2. Training and Inference

CNNs are deployed as a two step process: 1) training and 2) inference. Training is performed first with the result being either a continuous numerical value (regression) or a discrete class label (classification). Classification training involves applying a given annotated dataset as an input to the CNN, propagating it through the network, and comparing the output classification to the ground-truth label. The network weights are then updated typically using a backpropagation strategy such as Stochastic Gradient Descent (SGD) to reduce classification errors. This performs a search for the best weight values. Backpropogation is performed iteratively until a minimum acceptable error is reached or no further reduction in error is achieved. Backpropagation is compute intensive and traditionally performed in data centers that take advantage of dedicated GPUs or specialized training accelerators such as TPUs.

Fine-tuning is defined as retraining a previously trained model. It is easier to recover the accuracy of a quantized or pruned model with fine-tuning versus training from scratch.

CNN inference classification takes a previously trained classification model and predicts the class from input data not in the training dataset. Inference is not as computationally intensive as training and can be executed on edge, mobile, and embedded devices. The size of the inference network executing on mobile

Table 1
Acronyms and Abbreviations.

Acronym	Explanation
2D	Two Dimensional
3D	Three Dimensional
FP16	16-Bit Floating-Point
FP32	32-Bit Floating-Point
INT16	16-Bit Integer
INT8	8-Bit Integer
IR	Intermediate Representation
OFA	One-For-All
RGB	Red, Green, And Blue
SOTA	State of The Art
AI	Artificial Inteligence
BN	Batch Normalization
CBN	Conditional Batch Normalization
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EBP	Expectation Back Propagation
FCL	Fully Connected Layer
FCN	Fully Connected Networks
FLOP	Floating-Point Operation
GAP	Global Average Pooling
GEMM	General Matrix Multiply
GFLOP	Giga Floating-Point Operation
ILSVRC	Imagenet Large Visual Recognition Challenge
Im2col	Image To Column
KD	Knowledge Distillation
LRN	Local Response Normalization
LSTM	Long Short Term Memory
MAC	Multiply Accumulate
NAS	Network Architecture Search
NN	Neural Network
PTQ	Post Training Quantization
QAT	Quantization Aware Training
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
STE	Straight-Through Estimator
ASIC	Application Specific Integrated Circuit
AVX-512	Advance Vector Extension 512
CPU	Central Processing Unit
CU	Computing Unit
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HSA	Heterogeneous System Architecture
ISA	Instruction Set Architectures
PE	Processing Element
SIMD	Single Instruction Multiple Data
SoC	System on Chip
DPP	Determinantal Point Process
FFT	Fast Fourier Transfer
FMA	Fused Multiply–Add
KL-divergence	Kullback–Leibler Divergence
LASSO	Least Absolute Shrinkage And Selection Operator
MDP	Markov Decision Process
OLS	Ordinary Least Squares

devices may be limited due to memory, bandwidth, or processing constraints [79]. Pruning discussed in Section 3 and quantization discussed in Section 4 are two techniques that can alleviate these constraints.

In this paper, we focus on the acceleration of CNN inference classification. We compare techniques using standard benchmarks such as ImageNet [122], CIFAR [132], and MNIST [139]. The compression techniques are general and the choice of application domain doesn't restrict its use in object detection, natural language processing, etc.

2.3. Convolution Operations

The top of Fig. 2 shows a 3-channel image (e.g., RGB) as input to a convolutional layer. Because the input image has 3 channels, the convolution kernel must also have 3 channels. In this figure four $2 \times 2 \times 3$ convolution filters are shown, each consisting of three 2×2 kernels. Data is received from all 3 channels simultaneously. 12 image values are multiplied with the kernel weights producing a single output. The kernel is moved across the 3-channel image sharing the 12 weights. If the input image is $12 \times 12 \times 3$ the resulting output will be $11 \times 11 \times 1$ (using a stride of 1 and no padding). The filters work by extracting multiple smaller bit maps known as feature maps. If more filters are desired to learn different features they can be easily added. In this case 4 filters are shown resulting in 4 feature maps.

The standard convolution operation can be computed in parallel using a GEneral Matrix Multiply (GEMM) library [60]. Fig. 3 shows a parallel column approach. The 3D tensors are first flattened into 2D matrices. The resulting matrices are multiplied by the convolutional kernel which takes each input neuron (features), multiplies it, and generates output neurons (activations) for the next layer [138].

$$\mathbf{F}_n^{l+1} = \mathbf{A}_n^l = \text{activate}\left\{\sum_{m=1}^{M}\left(\mathbf{W}_{mn}^l * \mathbf{F}_m^l\right) + \mathbf{b}_n^l\right\} \tag{1}$$

Eq. 1 shows the layer-wise mathematical representation of the convolution layer where $\mathbf{W}$ represents the weights (filters) of the tensor with m input channels and n output channels, $\mathbf{b}$ represents the bias vector, and $\mathbf{F}^l$ represents the input feature tensor (typically from the activation of previous layer $\mathbf{A}^{l-1}$). $\mathbf{A}^l$ is the activated convolutional output. The goal of compression is to reduce the size of the $\mathbf{W}$ and $\mathbf{F}$ (or $\mathbf{A}$) without affecting accuracy.

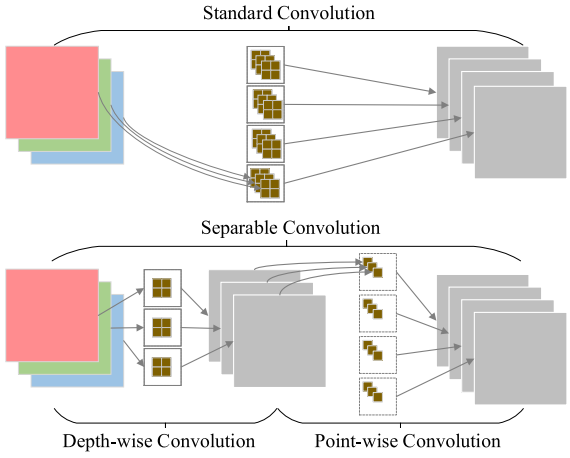


Fig. 2. Separable Convolution: A standard convolution is decomposed into depth-wise convolution and point-wise convolution to reduce both the model size and computations.

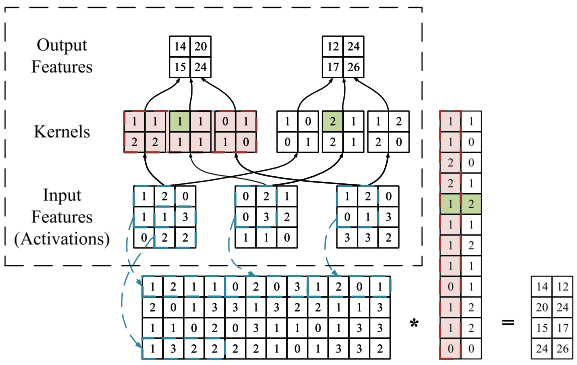


Fig. 3. Convolution Performance Optimization: From traditional convolution (dot squared) to image to column (im2col) - GEMM approach, adopted from [34]. The red and green boxes indicate filter-wise and shape-wise elements, respectively.

Fig. 4 shows a FCL - also called dense layer or dense connect. Every neuron is connected to each other neuron in a crossbar configuration requiring many weights. As an example, if the input and output channel are 1024 and 1000, respectively, the number of parameters in the filter will be a million by 1024×1000. As the image size grows or the number of features increase, the number of weights grows rapidly.

2.4. Efficient Structure

The bottom of Fig. 2 shows separable convolution implemented in MobileNet [105]. Separable convolution assembles a depth-wise convolution followed by a point-wise convolution. A depth-wise convolution groups the input feature by channel, and treats each channel as a single input tensor generating activations with the same number of channels. Point-wise convolution is a standard convolution with 1×1 kernels. It extracts mutual information across the channels with minimum computation overhead. For the $12 \times 12 \times 3$ image previously discussed, a standard convolution needs $2 \times 2 \times 3 \times 4$ multiplies to generate 1×1 outputs. Separable convolution needs only $2 \times 2 \times 3$ for depth-wise convolution and $1 \times 1 \times 3 \times 4$ for point-wise convolution. This reduces computations by half from 48 to 24. The number of weights is also reduced from 48 to 24.

The receptive field is the size of a feature map used in a convolutional kernel. To extract data with a large receptive filed and high precision, cascaded layers should be applied as in the top of Fig. 5. However, the number of computations can be reduced by expanding the network width with four types of filters as shown in Fig. 5. The concatenated result performs better than one convolutional layer with same computation workloads [226].

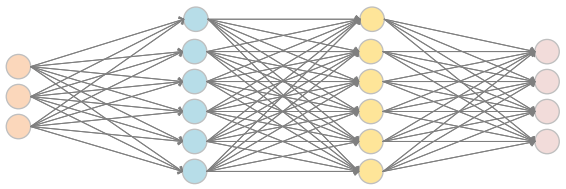


Fig. 4. Fully Connected Layer: Each node in a layer connects to all the nodes in the next layer, and every line corresponds to a weight value.

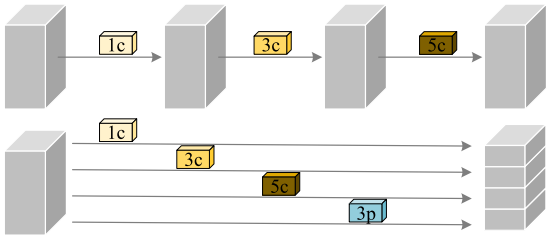


Fig. 5. Inception Block: The inception block computes multiple convolutions with one input tensor in parallel, which extends the receptive field by mixing the size of kernels. The yellow - brown coloured cubes are convolutional kernels sized 1, 3, and 5. The blue cube corresponds to a 3×3 pooling operation.

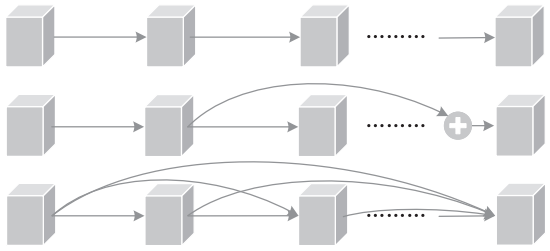


Fig. 6. Conventional Network Block (top), Residual Network Block (middle), and Densely Connected Network Block (bottom).

A residual network architecture block [98] is a feed forward layer with a short circuit between layers as shown in the middle of Fig. 6. The short circuit keeps information from the previous block to increase accuracy and avoid vanishing gradients during training. Residual networks help deep networks grow in depth by directly transferring information between deeper and shallower layers.

The bottom of Fig. 6 shows the densely connected convolutional block from DenseNets [109], this block extends both the network depth and the receptive field by delivering the feature of former layers to all the later layers in a dense block using concatenation. ResNets transfer outputs from a single previous layer. DenseNets build connections across layers to fully utilize previous features. This provides weight efficiencies.

2.5. Batch Normalization

BN was introduced in 2015 to speed up the training phase, and to improve the neural network performance [119]. Most SOTA neural networks apply BN after a convolutional layer. BN addresses internal covariate shift (an altering of the network activation distribution caused by modifications to parameters during training) by normalizing layer inputs. This has been shown to reduce training time up to 14×. Santurkar [210] argues that the efficiency of BN is from its ability to smooth values during optimization.

$$\mathbf{y} = \gamma \cdot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2}$$

Eq. 2 gives the formula for computing inference BN, where **x**and**y** are the input feature and the output of BN, γ and β are learned parameters, μ and σ are the mean value and standard deviation calculated from the training set, and ϵ is the additional small value (e.g., 1e-6) to prevent the denominator from being 0. The variables of Eq. 2 are determined in the training pass and integrated into the trained weights. If the features in one channel share the same parameters, then it turns to a linear transform on each output channel. Channel-wise BN parameters potentially helps channel-wise pruning. BN could also raise the performance of the cluster-based quantize technique by reducing parameter dependency [48].

Since the parameters of the BN operation are not modified in the inference phase, they may be combined with the trained weights and biases. This is called BN folding or BN merging. Eq. 3 show an example of BN folding. The new weight $\mathbf{W}'$ and bias $\mathbf{b}'$ are calculated using the pretrained weights **W** and BN parameters from Eq. 2. Since the new weight is computed after training and prior to inference, the number of multiplies are reduced and therefore BN folding decreases inference latency and computational complexity.

$$\mathbf{W}' = \gamma \cdot \frac{\mathbf{W}}{\sqrt{\sigma^2 + \epsilon}}, \quad \mathbf{b}' = \gamma \cdot \frac{\mathbf{b} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{3}$$

2.6. Pooling

Pooling was first published in the 1980s with neocognitron [71]. The technique takes a group of values and reduces them to a single value. The selection of the single replacement value can be computed as an average of the values (average pooling) or simply selecting the maximum value (max pooling).

Pooling destroys spatial information as it is a form of downsampling. The window size defines the area of values to be pooled. For image processing it is usually a square window with typical sizes being $2 \times 2, 3 \times 3$ or 4×4. Small windows allow enough information to be propagated to successive layers while reducing the total number of computations [224].

Global pooling is a technique where, instead of reducing a neighborhood of values, an entire feature map is reduced to a single value [154]. Global Average Pooling (GAP) extracts information from multi-channel features and can be used with dynamic pruning [153,42].

Capsule structures have been proposed as an alternative to pooling. Capsule networks replace the scalar neuron with vectors. The vectors represent a specific entity with more detailed information, such as position and size of an object. Capsule networks void loss of spatial information by capturing it in the vector representation. Rather than reducing a neighborhood of values to a single value, capsule networks perform a dynamic routing algorithm to remove connections [209].

2.7. Parameters

Fig. 7 show top-1 accuracy percent verses the number of operations needed for a number of popular neural networks [23]. The number of parameters in each network is represented by the size of the circle. A trend (not shown in the figure) is a yearly increase in parameter complexity. In 2012, AlexNet [133] was published with 60 million parameters. In 2013, VGG [217] was introduced with 133 million parameters and achieved 71.1% top-1 accuracy. These were part of the ImageNet large scale visual recognition challenge (ILSVRC) [207]. The competition's metric was top-1 absolute accuracy. Execution time was not a factor. This incentivized neural network designs with significant redundancy. As of 2020, models with more than 175 billion parameters have been published [26].

Networks that execute in data centers can accommodate models with a large number of parameters. In resource constrained

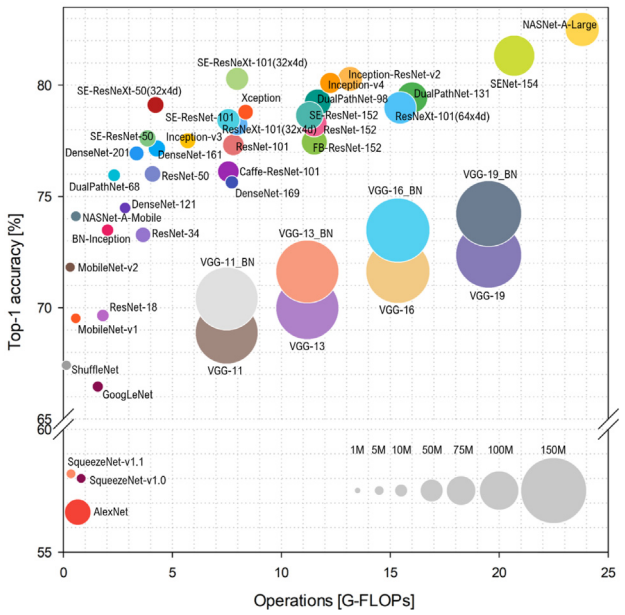


Fig. 7. Popular CNN Models: Top-1 accuracy vs GFLOPs and model size, adopted from [23].

environments such as edge and mobile deployments, reduced parameter models have been designed. For example, GoogLeNet [226] achieves similar top-1 accuracy of 69.78% as VGG-16 but with only 7 million parameters. MobileNet [105] has 70% top-1 accuracy with only 4.2 million parameters and only 1.14 Giga FLoating-point OPerations (GFLOPs). A more detailed network comparison can be found in [5].

3. Pruning

Network pruning is an important technique for both memory size and bandwidth reduction. In the early 1990s, pruning techniques were developed to reduce a trained large network into a smaller network without requiring retraining [201]. This allowed neural networks to be deployed in constrained environments such as embedded systems. Pruning removes redundant parameters or neurons that do not significantly contribute to the accuracy of results. This condition may arise when the weight coefficients are zero, close to zero, or are replicated. Pruning consequently reduces the computational complexity. If pruned networks are retrained it provides the possibility of escaping a previous local minima [43] and further improve accuracy.

Research on network pruning can roughly be categorized as sensitivity calculation and penalty-term methods [201]. Significant recent research interest has continued showing improvements for both network pruning categories or a further combination of them.

Recently, new network pruning techniques have been created. Modern pruning techniques may be classified by various aspects including: 1) *structured* and *unstructured pruning* depending if the pruned network is symmetric or not, 2) *neuron* and *connection pruning* depending on the pruned element type, or 3) *static* and *dynamic pruning*. Fig. 8 shows the processing differences between static and dynamic pruning. *Static pruning* has all pruning steps performed offline prior to inference while *dynamic pruning* is performed during runtime. While there is overlap between the categories, in this paper we will use *static pruning* and *dynamic pruning* for classification of network pruning techniques.

Fig. 9 shows a granularity of pruning opportunities. The four rectangles on the right side correspond to the four brown filters in the top of Fig. 2. Pruning can occur on an element-by-element, row-by-row, column-by-column, filter-by-filter, or layer-by-layer basis. Typically element-by-element has the smallest sparsity impact, and results in a unstructured model. Sparsity decreases from left-to-right in Fig. 9.

$$\begin{aligned} &\arg\min_{p} L = N(x;\mathbf{W}) - N_p(x;\mathbf{W}_p) \\ &\text{where } N_p(x;\mathbf{W}_p) = P(N(x;\mathbf{W})) \end{aligned} \tag{4}$$

Independent of categorization, pruning can be described mathematically as Eq. 4. N represents the entire neural network which contains a series of layers (e.g., convolutional layer, pooling layer, etc.) with x as input. L represents the pruned network with N_p performance loss compared to the unpruned network. Network performance is typically defined as accuracy in classification. The pruning function, $P(\cdot)$, results in a different network configuration N_p along with the pruned weights $\mathbf{W}_p$. The following sections are primarily concerned with the influence of $P(\cdot)$ on N_p. We also consider how to obtain $\mathbf{W}_p$.

3.1. Static Pruning

Static pruning is a network optimization technique that removes neurons offline from the network after training and before inference. During inference, no additional pruning of the network is performed. Static pruning commonly has three parts: 1) selection of parameters to prune, 2) the method of pruning the neurons, and 3) optionally fine-tuning or re-training [92]. Retraining may improve the performance of the pruned network to achieve comparable accuracy to the unpruned network but may require significant offline computation time and energy.

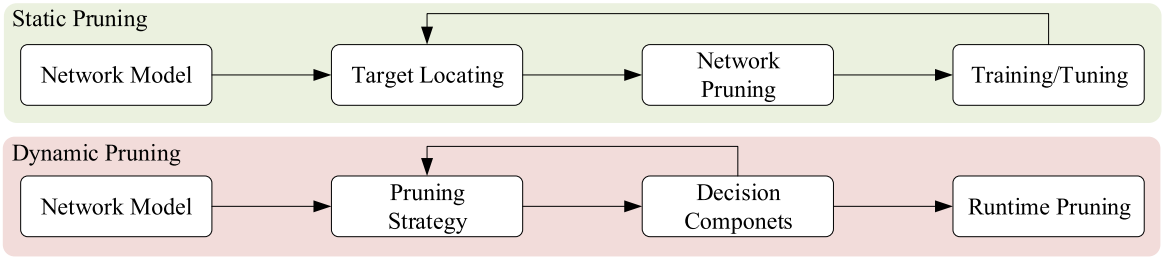


Fig. 8. Pruning Categories: *Static pruning* is performed offline prior to inference while *Dynamic pruning* is performed at runtime.

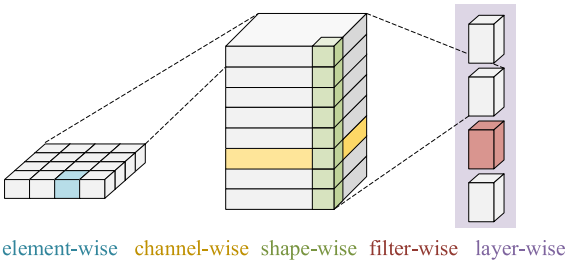


Fig. 9. Pruning Opportunities: Different network sparsity results from the granularity of pruned structures. Shape-wise pruning was proposed by Wen [241].

3.1.1. Pruning Criteria

As a result of network redundancy, neurons or connections can often be removed without significant loss of accuracy. As shown in Eq. 1, the core operation of a network is a convolution operation. It involves three parts: 1) input features as produced by the previous layer, 2) weights produced from the training phase, and 3) bias values produced from the training phase. The output of the convolution operation may result in either zero valued weights or features that lead to a zero output. Another possibility is that similar weights or features may be produced. These may be merged for distributive convolutions.

An early method to prune networks is brute-force pruning. In this method the entire network is traversed element-wise and weights that do not affect accuracy are removed. A disadvantage of this approach is the large solution space to traverse. A typical metric to determine which values to prune is given by the $l_p - \text{norm}, \text{s.t.} p \in \{N, \infty\}$, where N is natural number. The l_p-norm of a vector $\mathbf{x}$ which consists of n elements is mathematically described by Eq. 5.

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}} \tag{5}$$

Among the widely applied measurements, the l_1-norm is also known as the *Manhattan norm* and the l_2-norm is also known as the *Euclidean norm*. The corresponding l_1 and l_2 regularization have the names *LASSO* (least absolute shrinkage and selection operator) and *Ridge*, respectively [230]. The difference between the l_2-norm pruned tensor and an unpruned tensor is called the l_2-distance. Sometimes researchers also use the term l_0-norm defined as the total number of nonzero elements in a vector.

$$\begin{aligned} &\arg\min_{\alpha,\beta}\left\{\sum_{i=1}^{N}\left(y_i - \alpha - \sum_{j=1}^{p}\beta_j \mathbf{x}_{ij}\right)^2\right\} \\ &\text{subject to} \quad \sum_{j}^{p}|\beta_j| \leqslant t \end{aligned} \tag{6}$$

Equation Eq. 6 mathematically describes l_2 LASSO regularization. Consider a sample consisting of N cases, each of which consists of p covariates and a single outcome y_i. Let $x^i = (x_{i1}, \ldots, x_{ip})^T$ be the standardized covariate vector for the i-th case (input feature in DNNs), so we have $\sum_i x_{ij}/N = 0$, $\sum_i x_{ij}^2/N = 1$. β represents the coefficients $\beta = (\beta_1, \ldots, \beta_p)^T$ (weights) and t is a predefined tunning parameter that determines the sparsity. The LASSO estimate α is 0 when the average of y_i is 0 because for all t, the solution for α is $\alpha = \overline{y}$. If the constraint is $\sum_j^p \beta_j^2 \leqslant t$ then the Eq. 6 becomes Ridge regression. Removing the constraint will results in the Ordinary Least Squares (OLS) solution.

$$\arg\min_{\beta\in\mathbb{R}}\left\{\frac{1}{N}\|y - \mathbf{X}\beta\|_2^2 + \lambda\|\beta\|_1\right\} \tag{7}$$

Eq. 6 can be simplified into the so-called Lagrangian form shown in Eq. 7. The Lagrangian multiplier translates the objective function $f(x)$ and constraint $g(x) = 0$ into the format of $\mathscr{L}(x, \lambda) = f(x) - \lambda g(x)$, Where the $\|\cdot\|_p$ is the standard $l_p -$ norm, the $\mathbf{X}$ is the covariate matrix that contains x_{ij}, and λ is the data dependent parameter related to t from Eq. 6.

Both magnitude-based pruning and penalty based pruning may generate zero values or near-zero values for the weights. In this section we discuss both methods and their impact. Magnitude-based pruning: It has been proposed and is widely accepted that trained weights with large values are more important than trained weights with smaller values [143]. This observation is the key to magnitude-based methods. Magnitude-based pruning methods seek to identify unneeded weights or features to remove them from runtime evaluation. Unneeded values may be pruned either in the kernel or at the activation map. The most intuitive magnitude-based pruning methods is to prune all zero-valued weights or all weights within an absolute value threshold.

LeCun as far back as 1990 proposed Optimal Brain Damage (OBD) to prune single non-essential weights [140]. By using the second derivative (Hessian matrix) of the loss function, this static pruning technique reduced network parameters by a quarter. For a simplified derivative computation, OBD functions under three assumptions: 1) *quadratic* - the cost function is near-quadratic, 2) *extremal* - the pruning is done after the network converged, and 3) *diagonal* - sums up the error of individual weights by pruning the result of the error caused by their co-consequence. This research also suggested that the sparsity of DNNs could provide opportunities to accelerate network performance. Later Optimal Brain Surgeon (OBS) [97] extended OBD with a similar second-order method but removed the *diagonal* assumption in OBD. OBS considers the Hessian matrix is usually non-diagonal for most applications. OBS improved the neuron removal precision with up to a 90% reduction in weights for XOR networks.

These early methods reduced the number of connections based on the second derivative of the loss function. The training procedure did not consider future pruning but still resulted in networks that were amenable to pruning. They also suggested that methods based on Hessian pruning would exhibit higher accuracy than those pruned with only magnitude-based algorithms [97]. More recent DNNs exhibit larger weight values when compared to early DNNs. Early DNNs were also much shallower with orders of magnitude less neurons. GPT-3 [26], for example, contains 175-billion parameters while VGG-16 [217] contains just 133-million parameters. Calculating the Hessian matrix during training for networks with the complexity of GPT-3 is not currently feasible as it has the complexity of $O\left(W^2\right)$. Because of this simpler magnitude-based algorithms have been developed [177,141].

Filter-wise pruning [147] uses the l_1-norm to remove filters that do not affect the accuracy of the classification. Pruning entire filters and their related feature maps resulted in a reduced inference cost of 34% for VGG-16 and 38% for ResNet-110 on the CIFAR-10 dataset with improved accuracy 0.75% and 0.02%, respectively.

Most network pruning methods choose to measure weights rather than activations when rating the effectiveness of pruning [88]. However, activations may also be an indicator to prune corresponding weights. Average Percentage Of Zeros (APoZ) [106] was introduced to judge if one output activation map is contributing to the result. Certain activation functions, particularly rectification such as Rectified Linear Unit (ReLU), may result in a high percentage of zeros in activations and thus be amenable to pruning. Eq. 8 shows the definition of $\text{APoZ}_c^{(i)}$ of the c-th neuron in the i-th layer,

where $\mathbf{O}_c^{(i)}$ denotes the activation, N is the number of calibration (validation) images, and M is the dimension of activation map. $f(\text{true}) = 1$ and $f(\text{false}) = 0$.

$$\text{APoZ}_c^{(i)} = \text{APoZ}\left(\mathbf{O}_c^{(i)}\right) = \frac{\sum_{k=0}^{N}\sum_{j=0}^{M} f\left(\mathbf{O}_{cj}^{(i)}(k) = 0\right)}{N \times M} \tag{8}$$

Similarly, inbound pruning [195], also an activation technique, considers channels that do not contribute to the result. If the top activation channel in the standard convolution of Fig. 2 are determined to be less contributing, the corresponding channel of the filter in the bottom of the figure will be removed. After pruning this technique achieved about 1.5× compression.

Filter-wise pruning using a threshold from the sum of filters' absolute values can directly take advantage of the structure in the network. In this way, the ratio of pruned to unpruned neurons (i.e., the pruning ratio) is positively correlated to the percentage of kernel weights with zero values, which can be further improved by penalty-based methods. Penalty-based pruning: In penalty-based pruning, the goal is to modify an error function or add other constraints, known as bias terms, in the training process. A penalty value is used to update some weights to zero or near zero values. These values are then pruned.

Hanson [96] explored hyperbolic and exponential bias terms for pruning in the late 80s. This method uses weight decay in back-propagation to determine if a neuron should be pruned. Low-valued weights are replaced by zeros. Residual zero valued weights after training are then used to prune unneeded neurons.

Feature selection [55] is a technique that selects a subset of relevant features that contribute to the result. It is also known as attribute selection or variable selection. Feature selection helps algorithms avoiding over-fitting and accelerates both training and inference by removing features and/or connections that don't contribute to the results. Feature selection also aids model understanding by simplifying them to the most important features. Pruning in DNNs can be considered to be a kind of feature selection [123].

LASSO was previously introduced as a penalty term. LASSO shrinks the least absolute valued feature's corresponding weights. This increases weight sparsity. This operation is also referred to as LASSO feature selection and has been shown to perform better than traditional procedures such as OLS by selecting the most significantly contributed variables instead of using all the variables. This lead to approximately 60% more sparsity than OLS [181].

Element-wise pruning may result in an unstructured network organizations. This leads to sparse weight matrices that are not efficiently executed on instruction set processors. In addition they are usually hard to compress or accelerate without specialized hardware support [91]. Group LASSO [260] mitigates these inefficiencies by using a structured pruning method that removes entire groups of neurons while maintaining structure in the network organization [17].

Group LASSO is designed to ensure that all the variables sorted into one group could be either included or excluded as a whole. Eq. 9 gives the pruning constraint where $\mathbf{X}$ and β in Eq. 7 are replaced by the higher dimensional $\mathbf{X_j}$ and β_j for the j groups.

$$arg \min_{\beta \in \mathbb{R}^p} \left\{ \left\| y - \sum_{j=1}^{J} \mathbf{X}_j \beta_j \right\|_2^2 + \lambda \sum_{j=1}^{J} \left\| \beta_j \right\|_{K_j} \right\} \tag{9}$$

Fig. 10 shows Group LASSO with group shapes used in Structured Sparsity Learning (SSL) [241]. Weights are split into multiple groups. Unneeded groups of weights are removed using LASSO feature selection. Groups may be determined based on geometry, computational complexity, group sparsity, etc. SSL describes an example where group sparsity in row and column directions may be used to reduce the execution time of GEMM. SSL has shown improved inference times on AlexNet with both CPUs and GPUs by 5.1× and 3.1×, respectively.

Group-wise brain damage [136] also introduced the group LASSO constraint but applied it to filters. This simulates brain damage and introduces sparsity. It achieved 2× speedup with 0.7% ILSVRC-2012 accuracy loss on the VGG Network.

Sparse Convolutional Neural Networks (SCNN) [17] take advantage of two-stage tensor decomposition. By decomposing the input feature map and convolutional kernels, the tensors are transformed into two tensor multiplications. Group LASSO is then applied. SCNN also proposed a hardware friendly algorithm to further accelerate sparse matrix computations. They achieved 2.47× to 6.88× speed-up on various types of convolution.

Network slimming [158] applies LASSO on the scaling factors of BN. BN normalizes the activation by statistical parameters which are obtained during the training phase. Network slimming has the effect of introducing forward invisible additional parameters without additional overhead. Specifically, by setting the BN scaler parameter to zero, channel-wise pruning is enabled. They achieved 82.5% size reduction with VGG and 30.4% computation compression without loss of accuracy on ILSVRC-2012.

Sparse structure selection [111] is a generalized network slimming method. It prunes by applying LASSO to sparse scaling factors in neurons, groups, or residual blocks. Using an improved gradient method, Accelerated Proximal Gradient (APG), the proposed method shows better performance without fine-tunning achieving 4× speed-up on VGG-16 with 3.93% ILSVRC-2012 top-1 accuracy loss.

Dropout: While not specifically a technique to prune networks, dropout does reduce the number of parameters [222]. It was originally designed as a stochastic regularizer to avoid over-fitting of data [103]. The technique randomly omits a percentage of neurons typically up to 50%, This *dropout* operation breaks off part of the connections between neurons to avoid co-adaptations. Dropout could also be regarded as an operation that separately trains many sub-networks and takes the average of them during the inference phase. Dropout increases training overhead but it does not affect the inference time.

Sparse variational dropout [176] added a dropout hyperparameter called the dropout rate to reduce the weights of VGG-like networks by 68×. During training the dropout rate can be used to identify single weights to prune. This can also be applied with other compression approaches for further reduction in weights.

Redundancies: The goal of norm-based pruning algorithms is to remove zeros. This implies that the distribution of values should wide enough to retain some values but contain enough values close to zero such that a smaller network organization is still accurate. This does not hold in some circumstances. For example, filters that have small norm deviations or a large minimum norm have small search spaces making it difficult to prune based on a threshold [100]. Even when parameter values are wide enough, in some networks smaller values may still play an important role in pro-

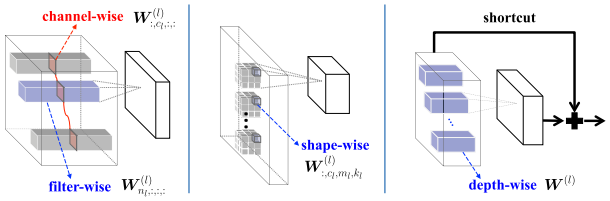


Fig. 10. Types of Sparsity Geometry, adopted from [241].

ducing results. One example of this is when large valued parameters saturate [64]. In these cases magnitude-based pruning of zero values may decrease result accuracy.

Similarly, penalty-based pruning may cause network accuracy loss. In this case, the filters identified as unneeded due to similar coefficient values in other filters may actually be required. Removing them may significantly decrease network accuracy [88]. Section 3.1.2 describes techniques to undo pruning by tuning the weights to minimize network loss while this section describes redundancy based pruning.

Using BN parameters, feature map channel distances can be computed by layer [266]. Using a clustering approach for distance, nearby features can be tuned. An advantage of clustering is that redundancy is not measured with an absolute distance but a relative value. With about 60 epochs of training they were able to prune the network resulting in a 50% reduction in FLOPs (including non-convolutional operations) with a reduction in accuracy of only 1% for both top-1 and top-5 on the ImageNet dataset.

Filter pruning via geometric median (FPGM) [100] identifies filters to prune by measuring the l_2-distance using the geometric median. FPGM found 42% FLOPs reduction with 0.05% top-1 accuracy drop on ILSVRC-2012 with ResNet-101.

The reduce and reused (also described as outbound) [4] method [195] prunes entire filters by computing the statistical variance of each filter's output using a calibration set. Filters with low variance are pruned. The outbound method obtained 2.37× acceleration with 1.52% accuracy loss on Labeled Faces in the Wild (LFW) dataset [110] in the filed of face recognition.

A method that iteratively removes redundant neurons for FCLs without requiring special validation data is proposed in [221]. This approach measures the similarity of weight groups after a normalization. It removes redundant weights and merges the weights into a single value. This lead to a 34.89% reduction of FCL weights on AlexNet with 2.24% top-1 accuracy loss on ILSVRC-2012.

Comparing with the similarity based approach above, DIVersity NETworks (DIVNET) [167] considers the calculation redundancy based on the activations. DIVNET introduces Determinantal Point Process (DPP) [166] as a pruning tool. DPP sorts neurons into categories including dropped and retained. Instead of forcing the removal of elements with low contribution factors, they fuse the neurons by a process named re-weighting. Re-weighting works by minimizing the impact of neuron removal. This minimizes pruning influence and mitigates network information loss. They found 3% loss on CIFAR-10 dataset when compressing the network into half weight.

ThiNet [164] adopts statistics information from the next layer to determine the importance of filters. It uses a greedy search to prune the channel that has the smallest reconstruction cost in the next layer. ThiNet prunes layer-by-layer instead of globally to minimize large errors in classification accuracy. It also prunes less during each training epoch to allow for coefficient stability. The pruning ratio is a predefined hyper-parameter and the runtime complexity is directly related to the pruning ratio. ThiNet compressed ResNet-50 FLOPs to 44.17% with a top-1 accuracy reduction of 1.87%.

He [101] adopts LASSO regression instead of a greedy algorithm to estimate the channels. Specifically, in one iteration, the first step is to evaluate the most important channel using the l_1-norm. The next step is to prune the corresponding channel that has the smallest Mean Square Error (MSE). Compared to an unpruned network, this approach obtained 2× acceleration of ResNet-50 on ILSVRC-2012 with about 1.4% accuracy loss on top-5, and a 4× reduction in execution time with top-5 accuracy loss of 1.0% for VGG-16. The authors categorize their approach as dynamic inference-time channel pruning. However it requires 5000 images for calibration with 10 samples per image and more importantly results in a statically pruned network. Thus we have placed it under static pruning.

3.1.2. Pruning combined with Tuning or Retraining

Pruning removes network redundancies and has the benefit of reducing the number of computations without significant impact on accuracy for some network architectures. However, as the estimation criterion is not always accurate, some important elements may be eliminated resulting in a decrease in accuracy. Because of the loss of accuracy, time-consuming fine-tuning or re-training may be employed to increase accuracy [258].

Deep compression [92], for example, describes a static method to prune connections that don't contribute to classification accuracy. In addition to feature map pruning they also remove weights with small values. After pruning they re-train the network to improve accuracy. This process is performed iteratively three times resulting in a 9× to 13× reduction in total parameters with no loss of accuracy. Most of the removed parameters were from FCLs.

Recoverable Pruning: Pruned elements usually cannot be recovered. This may result in reduced network capability. Recovering lost network capability requires significant re-training. Deep compression required millions of iterations to retrain the network [92]. To avoid this shortcoming, many approaches adopt recoverable pruning algorithms. The pruned elements may also be involved in the subsequent training process and adjust themselves to fit the pruned network.

Guo [88] describes a recoverable pruning method using binary mask matrices to indicate whether a single weight value is pruned or not. The l_1-norm pruned weights can be stochastically spliced back into the network. Using this approach AlexNet was able to be reduced by a factor of 17.7× with no accuracy loss. Retraining iterations were significantly reduced to 14.58% of Deep compression [92]. However this type of pruning still results in an asymmetric network complicating hardware implementation.

Soft Filter Pruning (SFP) [99] further extended recoverable pruning using a dimension of filter. SFP obtained structured compression results with an additional benefit or reduced inference time. Furthermore, SFP can be used on difficult to compress networks achieving a 29.8% speed-up on ResNet-50 with 1.54% ILSVRC-2012 top-1 accuracy loss. Comparing with Guo's recoverable weight [88] technique, SFP achieves inference speed-ups closer to theoretical results on general purpose hardware by taking advantage of the structure of the filter.

Increasing Sparsity: Another motivation to apply fine-tuning is to increase network sparsity. Sparse constraints [270] applied low rank tensor constraints [157] and group sparsity [57] achieving a 70% reduction of neurons with a 0.57% drop of AlexNet in ILSVRC-2012 top-1 accuracy.

Adaptive Sparsity: No matter what kind of pruning criteria is applied, a layer-wise pruning ratio usually requires a human decision. Too high a ratio resulting in very high sparsity may cause the network to diverge requiring heavy re-tuning.

Network slimming [158], previously discussed, addresses this problem by automatically computing layer-wise sparsity. This achieved a 20× model size compression, 5× computing reduction, and less than 0.1% accuracy loss on the VGG network.

Pruning can also be performed using a min–max optimization module [218] that maintains network accuracy during tuning by keeping a pruning ratio. This technique compressed the VGG network by a factor of 17.5× and resulted in a theoretical execution time (FLOPs) of 15.56% of the unpruned network. A similar approach was proposed with an estimation of weights sets [33]. By avoiding the use of a greedy search to keep the best pruning ratio, they achieved the same ResNet classification accuracy with only 5% to 10% size of original weights.

AutoPruner [163] integrated the pruning and fine-tuning of a three-stage pipeline as an independent training-friendly layer. The layer helped gradually prune during training eventually resulting in a less complex network. AutoPruner pruned 73.59% of compute operations on VGG-16 with 2.39% ILSVRC-2012 top-1 loss. ResNet-50 resulted in a 65.80% of compute operations with 3.10% loss of accuracy.

Training from Scratch: Observation shows that network training efficiency and accuracy is inversely proportional to structure sparsity. The more dense the network, the less training time [94,147,70]. This is one reason that current pruning techniques tend to follow a train-prune-tune pipeline rather than training a pruned structure from scratch.

However, the lottery ticket hypothesis [70] shows that it is not of primary importance to preserve the original weights but the initialization. Experiments show that dense, randomly-initialized pruned sub-networks can be trained effectively and reach comparable accuracy to the original network with the same number of training iterations. Furthermore, standard pruning techniques can uncover the aforementioned sub-networks from a large oversized network - the *Winning Tickets*. In contrast with current static pruning techniques, the lottery ticket hypothesis after a period of time drops all well-trained weights and resets them to an initial random state. This technique found that ResNet-18 could maintain comparable performance with a pruning ratio up to 88.2% on the CIFAR-10 dataset.

Towards Better Accuracy: By reducing the number of network parameters, pruning techniques can also help to reduce overfitting. Dense-Sparse-Dense (DSD) training [93] helps various network improve classification accuracy by 1.1% to 4.3%. DSD uses a three stage pipeline: 1) dense training to identify important connections, 2) prune insignificant weights and sparse training with a sparsity constraint to take reduce the number of parameters, and 3) re-dense the structure to recover the original symmetric structure, this also increase the model capacity. The DSD approach has also shown impressive performance on the other type of deep networks such as Recurrent Neural Networks (RNNs) and Long Short Term Memory networks (LSTMs).

3.2. Dynamic Pruning

Except for recoverable techniques, static pruning permanently destroys the original network structure which may lead to a decrease in model capability. Techniques have been researched to recover lost network capabilities but once pruned and retrained, the static pruning approach can't recover destroyed information. Additionally, observations shows that the importance of neuron binding is input-independent [73].

Dynamic pruning determines at runtime which layers, channels, or neurons will not participate in further activity. Dynamic pruning can overcome limitations of static pruning by taking advantage of changing input data potentially reducing computation, bandwidth, and power dissipation. Dynamic pruning typically doesn't perform runtime fine-tuning or re-training. In Fig. 11, we show an overview of dynamic pruning systems. The most important consideration is the decision system that decides what to prune. The related issues are:

1. The type of the decision components: a) additional connections attached to the original network used during the inference phase and/or the training phase, b) characteristics of the connections that can be learned by standard backpropagation algorithms [73], and c) a side decision network which tends to perform well but is often difficult to train [153].
2. The pruning level (shape): a) channel-wise [153,73,42], b) layer-wise [145], c) block-wise [246], or d) network-wise [25]. The pruning level chosen influences hardware design.
3. Input data: a) one-shot information feeding [246] feeds the entire input to the decision system, and b) layer-wise information feeding [25,68] where a window of data is iteratively fed to the decision system along with the forwarding.
4. Computing a decision score: l_p-norm [73], or b) other approaches [108].

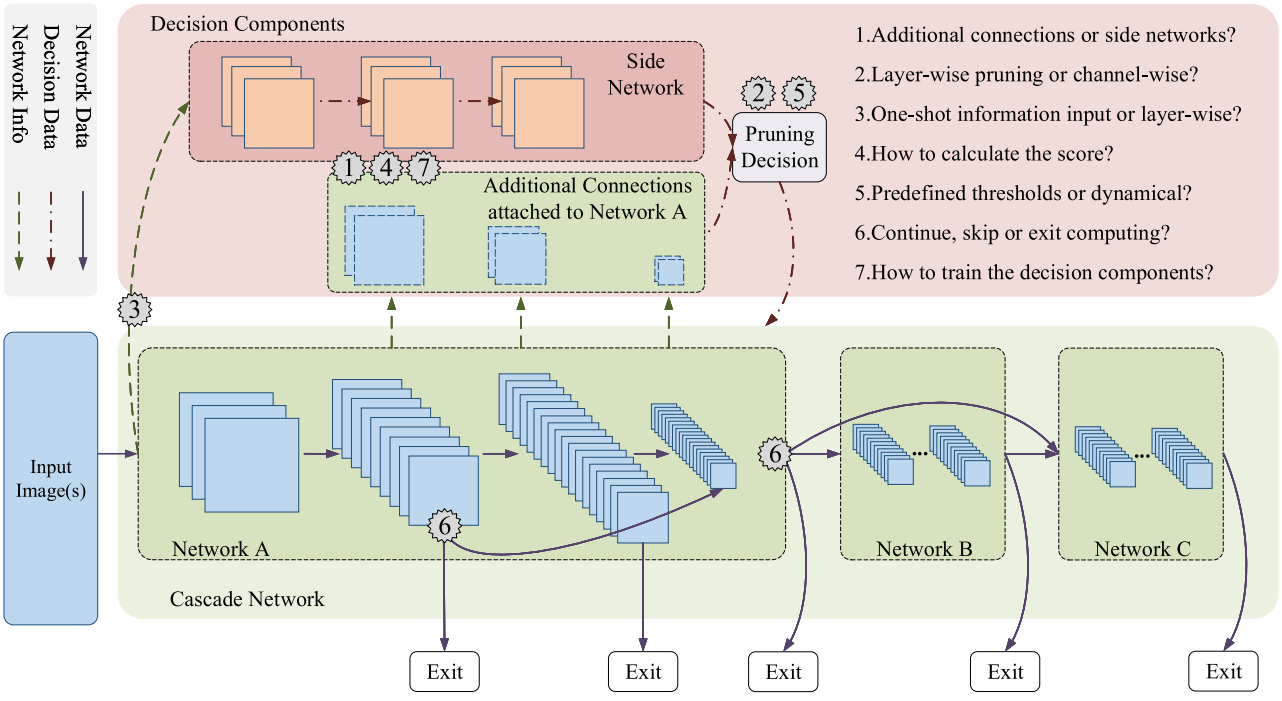


Fig. 11. Dynamic Pruning System Considerations.

5. Score comparison: a) human experience/experiment results [145] or b) automatic threshold or dynamic mechanisms [108].
6. Stopping criteria: a) in the case of layer-wise and network-wise pruning, some pruning algorithms skip the pruned layer/network [19,246], b) some algorithms dynamically choose the data path [189,259], and c) ending the computation and outputing the predicting results [68,145,148]. In this case the remaining layers are considered to be pruned.
7. Training the decision component: a) attached connections can be trained along with the original network [145,148,73], b) side networks are typically trained using reinforcement learning (RL) algorithms [19,153,189,246].

For instruction set processors, feature maps or the number of filters used to identify objects is a large portion of bandwidth usage [225] - especially for depth-wise or point-wise convolutions where features consume a larger portion of the bandwidth [47]. Dynamic tuning may also be applied to statically pruned networks potentially further reducing compute and bandwidth requirements.

A drawback of dynamic pruning is that the criteria to determine which elements to prune must be computed at runtime. This adds overhead to the system requiring additional compute, bandwidth, and power. A trade-off between dynamic pruning overhead, reduced network computation, and accuracy loss, should be considered. One method to mitigate power consumption inhibits computations from 0-valued parameters within a Processing Element (PE) [153].

3.2.1. Conditional Computing

Conditional computing involves activating an optimal part of a network without activating the entire network. Non-activated neurons are considered to be pruned. They do not participate in the result thereby reducing the number of computations required. Conditional computing applies to training and inference [20,56].

Conditional computing has a similarity with RL in that they both learn a pattern to achieve a reward. Bengio [19] split the network into several blocks and formulates the block chosen policies as an RL problem. This approach consists of only fully connected neural networks and achieved a 5.3× speed-up on CIFAR-10 dataset without loss of accuracy.

3.2.2. Reinforcement Learning Adaptive Networks

Adaptive networks aim to accelerating network inference by conditionally determining early exits. A trade-off between network accuracy and computation can be applied using thresholds. Adaptive networks have multiple intermediate classifiers to provide the ability of an early exit. A cascade network is a type of adaptive network. Cascade networks are the combinations of serial networks which all have output layers rather than per-layer outputs. Cascade networks have a natural advantage of an early exit by not requiring all output layers to be computed. If the early accuracy of a cascade network is not sufficient, inference could potentially be dispatched to a cloud device [145,25]. A disadvantage of adaptive networks is that they usually need hyper-parameters optimized manually (e.g., confidence score [145]). This introduces automation challenges as well as classification accuracy loss. They found 28.75% test error on CIFAR-10 when setting the threshold to 0.5. A threshold of 0.99 lowered the error to 15.74% at a cost of 3x to inference time.

A cascading network [189] is an adaptive network with an RL trained *Composer* that can determine a reasonable computation graph for each input. An adaptive controller *Policy Preferences* is used to intelligently enhance the *Composer* allowing an adjustment of the network computation graph from sub-graphs. The *Composer* performs much better in terms of accuracy than the baseline network with the same number of computation-involved parameters on a modified dataset, namely Wide-MNIST. For example, when invoking 1k parameters, the baseline achieves 72% accuracy while the *Composer* obtained 85%.

BlockDrop [246] introduced a *policy network* that [3] trained using RL to make an image-specific determination whether a residual network block should participate in the following computation. While the other approaches compute an exit confidence score per layer, the *policy network* runs only once when an image is loaded. It generates a boolean vector that indicates which residual blocks are activate or inactive. BlockDrop adds more flexibility to the early exit mechanism by allowing a decision to be made on any block and not just early blocks in Spatially Adaptive Computation Time (SACT) [68]. This is discussed further in Section 3.2.3. BlockDrop achieves an average speed-up of 20% on ResNet-101 for ILSVRC-2012 without accuracy loss. Experiments using the CIFAR dataset showed better performance than other SOTA counterparts at that time [68,82,147].

Runtime Neural Pruning (RNP) [153] is a framework that prunes neural networks dynamically. RNP formulates the feature selection problem as a Markov Decision Process (MDP) and then trains an RNN-based decision network by RL. The MDP reward function in the state-action-reward sequence is computation efficiency. Rather than removing layers, a side network of RNP predicts which feature maps are not needed. They found 2.3× to 5.9× reduction in execution time with top-5 accuracy loss from 2.32% to 4.89% for VGG-16.

3.2.3. Differentiable Adaptive Networks

Most of the aforementioned decision components are non-differential, thus computationally expensive RL is adopted for training. A number of techniques have been developed to reduce training complexity by using differentiable methods.

Dynamic channel pruning [73] proposes a method to dynamically select which channel to skip or to process using Feature Boosting and Suppression (FBS). FBS is a side network that guides channel amplification and omission. FBS is trained along with convolutional networks using SGD with LASSO constraints. The selecting indicator can be merged into BN parameters. FBS achieved 5× acceleration on VGG-16 with 0.59% ILSVRC-2012 top-5 accuracy loss, and 2× acceleration on ResNet-18 with 2.54% top-1, 1.46% top-5 accuracy loss.

Another approach, Dynamic Channel Pruning (DCP) [42] dynamically prunes channels using a *channel threshold weighting (T-Weighting)* decision. Specifically, this module prunes the channels whose score is lower than a given threshold. The score is calculated by a T-sigmoid activation function, which is mathematically described in Eq. 10, where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. The input to the T-sigmoid activation function is down sampled by a FCL from the feature maps. The threshold is found using iterative training which can be a computationally expensive process. DCP increased VGG-16 top-5 error by 4.77% on ILSVRC-2012 for 5× computation speed-up. By comparison, RNP increased VGG-16 top-5 error by 4.89% [153].

$$h(x) = \begin{cases} \sigma(x), & \text{if } x > T \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The cascading neural network by Leroux [145] reduced the average inference time of overfeat network [211] by 40% with a 2% ILSVRC-2012 top-1 accuracy loss. Their criteria for early exit is based on the confidence score generated by an output layer. The auxiliary layers were trained with general backpropagation. The adjustable score threshold provides a trade-off between accuracy and efficiency.

Bolukbasi [25] reports a system that contains a combination of other SOTA networks (e.g., AlexNet, ResNet, GoogLeNet, etc.). A policy adaptively chooses a point to exit early. This policy can be trained by minimizing its cost function. They format the system as a directed acyclic graph with various pre-trained networks as

basic components. They evaluate this graph to determine leaf nodes for early exit. The cascade of acyclic graphs with a combination of various networks reduces computations while maintaining prediction accuracy. ILSVRC-2012 experiments show ResNet-50 acceleration of 2.8× with 1% top-5 accuracy loss and 1.9× speed-up with no accuracy loss.

Considering the similarity of RNNs and residual networks [83], Spatially Adaptive Computation Time (SACT) [68] explored an early stop mechanism of residual networks in the spatial domain. SACT can be applied to various tasks including image classification, object detection, and image segmentation. SACT achieved about 20% acceleration with no accuracy loss for ResNet-101 on ILSVRC-2012.

To meet the computation constraints, Multi-Scale Dense Networks (MSDNets) [108] designed an adaptive network using two techniques: 1) an *anytime-prediction* to generate prediction results at many nodes to facilitate the network's early exit and 2) *batch computational budget* to enforce a simpler exit criteria such as a computation limit. MSDNets combine multi-scale feature maps [265] and dense connectivity [109] to enable accurate early exit while maintaining higher accuracy. The classifiers are differentiable so that MSDNets can be trained using stochastic gradient descent. MSDNets achieve 2.2× speed-up at the same accuracy for ResNet-50 on ILSVRC-2012 dataset.

To address the training complexity of adaptive networks, Li [148] proposed two methods. The first method is gradient equilibrium (GE). This technique helps backbone networks converge by using multiple intermediate classifiers across multiple different network layers. This improves the gradient imbalance issue found in MSDNets [108]. The second method is an Inline Subnetwork Collaboration (ISC) and a One-For-All knowledge distillation (OFA). Instead of independently training different exits, ISC takes early predictions into later predictors to enhance their input information. OFA supervises all the intermediate exits using a final classifier. At a same ILSVRC-2012 top-1 accuracy of 73.1%, their network takes only one-third the computational budget of ResNet.

Slimmable Neural Networks (SNN) [259] are a type of networks that can be executed at different widths. Also known as switchable networks, the network enables dynamically selecting network architectures (width) without much computation overhead. Switchable networks are designed to adaptively and efficiently make trade-offs between accuracy and on-device inference latency across different hardware platforms. SNN found that the difference of feature mean and variance may lead to training faults. SNN solves this issue with a novel switchable BN technique and then trains a wide enough network. Unlike cascade networks which primarily benefit from specific blocks, SNN can be applied with many more types of operations. As BN already has two parameters as mentioned in Section 2, the network switch that controls the network width comes with little additional cost. SNN increased top-1 error by 1.4% on ILSVRC-2012 while achieving about 2× speed-up.

3.3. Comparisons

Pruning techniques are diverse and difficult to compare. Shrinkbench [24] is a unified benchmark framework aiming to provide pruning performance comparisons.

There exist ambiguities about the value of the pre-trained weights. Liu [160] argues that the pruned model could be trained from scratch using a random weight initialization. This implies the pruned architecture itself is crucial to success. By this observation, the pruning algorithms could be seen as a type of NAS. Liu concluded that because the weight values can be re-trained, by themselves they are not efficacious. However, the lottery ticket hypothesis [70] achieved comparable accuracy only when the weight initialization was exactly the same as the unpruned model. Glae [72] resolved the discrepancy by showing that what really matters is the pruning form. Specifically, unstructured pruning can only be fine-tuned to restore accuracy but structured pruning can be trained from scratch. In addition, they explored the performance of dropout and l_0 regularization. The results showed that simple magnitude based pruning can perform better. They developed a magnitude based pruning algorithm and showed the pruned ResNet-50 obtained higher accuracy than SOTA at the same computational complexity.

4. Quantization

Quantization is known as *the process of approximating a continuous signal by a set of discrete symbols or integer values*. Clustering and parameter sharing also fall within this definition [92]. Partial quantization uses clustering algorithms such as k-means to quantize weight states and then store the parameters in a compressed file. The weights can be decompressed using either a lookup table or a linear transformation. This is typically performed during runtime inference. This scheme only reduces the storage cost of a model. This is discussed in Section 4.2.4. In this section we focus on numerical low-bit quantization.

Compressing CNNs by reducing precision values has been previously proposed. Converting floating-point parameters into low numerical precision datatypes for quantizing neural networks was proposed as far back as the 1990s [67,14]. Renewed interest in quantization began in the 2010s when 8-bit weight values were shown to accelerate inference without a significant drop in accuracy [233].

Historically most networks are trained using FP32 numbers [225]. For many networks an FP32 representation has greater precision than needed. Converting FP32 parameters to lower bit representations can significantly reduce bandwidth, energy, and on-chip area.

Fig. 12 shows the evolution of quantization techniques. Initially, only weights were quantized. By quantizing, clustering, and sharing, weight storage requirements can be reduced by nearly 4×. Han [92] combined these techniques to reduce weight storage requirements from 27 MB to 6.9 MB. Post training quantization involves taking a trained model, quantizing the weights, and then re-optimizing the model to generate a quantized model with scales [16]. Quantization-aware training involves fine-tuning a stable full precision model or re-training the quantized model. During this process real-valued weights are often down-scaled to integer values - typically 8-bits [120]. Saturated quantization can be used to generate feature scales using a calibratation algorithm with a calibration set. Quantized activations show similar distributions with previous real-valued data [173]. Kullback–Leibler divergence (KL-divergence, also known as relative entropy or information divergence) calibrated quantization is typically applied and can accelerate the network without accuracy loss for many well known models [173]. Fine-tuning can also be applied with this approach.

KL-divergence is a measure to show the relative entropy of probability distributions between two sets. Eq. 11 gives the equation for KL-divergence. P and Q are defined as discrete probability distributions on the same probability space. Specifically, P is the original data (floating-point) distribution that falls in several bins. Q is the quantized data histogram.

$$D_{\mathrm{KL}}(P\|Q) = \sum_{i=0}^{N} P(x_i) \log\left(\frac{P(x_i)}{Q(x_i)}\right) \tag{11}$$

Depending upon the processor and execution environment, quantized parameters can often accelerate neural network inference.

Quantization research can be categorized into two focus areas: 1) quantization aware training (QAT) and 2) post training quantiza-

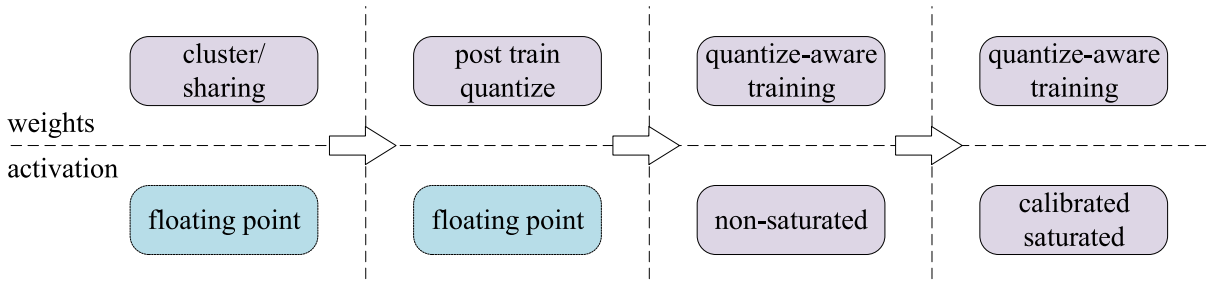


Fig. 12. Quantization Evolution: The development of quantization techniques, from left to right. Purple rectangles indicated quantized data while blue rectangles represent full precision 32-bit floating point format.

tion (PTQ). The difference depends on whether training progress is taken into account during training. Alternatively, we could also categorize quantization by where data is grouped for quantization: 1) layer-wise and 2) channel-wise. Further, while evaluating parameter widths, we could further classify by length: N-bit quantization.

Reduced precision techniques do not always achieve the expected speedup. For example, INT8 inference doesn't [4] achieve exactly 4× speedup over 32-bit floating point due to the additional operations of quantization and dequantization. For instance, Google's TensorFlow-Lite [227] and nVidia's Tensor RT [173] INT8 inference speedup is about $2-3\times$. Batch size is the capability to process more than one image in the forward pass. Using larger batch sizes, Tensor RT does achieve $3-4\times$ acceleration with INT8 [173].

Section 8 summarizes current quantization techniques used on the ILSVRC-2012 dataset along with their bit-widths for weights and activation.

4.1. Quantization Algebra

$$\mathbf{X}_q = f(s \times g(\mathbf{X}_r) + z) \tag{12}$$

There are many methods to quantize a given network. Generally, they are formulated as Eq. 12 where s is a scalar that can be calculated using various methods. $g(\cdot)$ is the clamp function applied to floating-point values $\mathbf{X}_r$ performing the quantization. z is the zero-point to adjust the true zero in some asymmetrical quantization approaches. $f(\cdot)$ is the rounding function. This section introduces quantization using the mathematical framework of Eq. 12.

$$clamp(x, \alpha, \beta) = max(min(x, \beta), \alpha) \tag{13}$$

Eq. 13 defines a clamp function. The *min–max* [4] method is given by Eq. 14 where $[m, M]$ are the bounds for the minimum and maximum values of the parameters, respectively. n is the maximum representable number derived from the bit-width (e.g., $256 = 2^8$ in case of 8-bit), and z, s are the same as in Eq. 12. z is typically non-zero in the *min–max* method [120].

$$\begin{aligned} &g(x) = clamp(x, m, M) \\ &s = \tfrac{n-1}{M-m},\ z = \tfrac{m \times (1-n)}{M-m} \\ &\text{where } m = \min\{\mathbf{X}_i\},\ M = \max\{\mathbf{X}_i\} \end{aligned} \tag{14}$$

The *max-abs* method uses a symmetry bound shown in Eq. 15. The quantization scale s is calculated from the largest one R among the data to be quantized. Since the bound is symmetrical, the zero point z will be zero. In such a situation, the overhead of computing an offset-involved convolution will be reduced but the dynamic range is reduced since the valid range is narrower. This is especially noticeable for ReLU activated data where all of which values fall on the positive axis.

$$\begin{aligned} &g(x) = clamp(x, -M, M) \\ &s = \tfrac{n-1}{R},\ z = 0 \\ &\text{where } R = \max\{abs\{\mathbf{X}_i\}\} \end{aligned} \tag{15}$$

Quantization can be applied on input features **F**, weights **W**, and biases **b**. Taking feature **F** and weights **W** as an example (ignoring the biases) and using the *min–max* method gives Eq. 16. The subscripts *r*and*q* denote the real-valued and quantized data, respectively. The *max* suffix is from R in Eq. 15, while $s_f = (n-1)/F_{max},\ s_w = (n-1)/W_{max}$.

$$\mathbf{F}_q = \frac{n-1}{F_{max}} \times \mathbf{F}_r, \quad \mathbf{W}_q = \frac{n-1}{W_{max}} \times \mathbf{W}_r \tag{16}$$

Integer quantized convolution is shown in Eq. 17 and follows the same form as convolution with real values. In Eq. 17, the $*$ denotes the convolution operation, **F** the feature, **W** the weights, and $\mathbf{O}_q$, the quantized convolution result. Numerous third party libraries support this type of integer quantized convolution acceleration. They are discussed in Section 4.3.2.

$$\mathbf{O}_q = \mathbf{F}_q * \mathbf{W}_q \quad \text{s.t. } \mathbf{F}, \mathbf{W} \in \mathbb{Z} \tag{17}$$

De-quantizing converts the quantized value $\mathbf{O}_q$ back to floating-point $\mathbf{O}_r$ using the feature scales s_f and weights scales s_w. A symmetric example with $z = 0$ is shown in Eq. 18. This is useful for layers that process floating-point tensors. Quantization libraries are discussed in Section 4.3.2.

$$\mathbf{O}_r = \frac{\mathbf{O}_q}{s_f \times s_w} = \mathbf{O}_q \times \frac{F_{max}}{(n-1)} \times \frac{W_{max}}{(n-1)} \tag{18}$$

In most circumstances, consecutive layers can compute with quantized parameters. This allows dequantization to be merged in one operation as in Eq. 19. $\mathbf{F}_q^{l+1}$ is the quantized feature for next layer and s_f^{l+1} is the feature scale for next layer.

$$\mathbf{F}_q^{l+1} = \frac{\mathbf{O}_q \times s_f^{l+1}}{s_f \times s_w} \tag{19}$$

The activation function can be placed following either the quantized output $\mathbf{O}_q$, the de-quantized output $\mathbf{O}_r$, or after a re-quantized output $\mathbf{F}_q^{l+1}$. The different locations may lead to different numerical outcomes since they typically have different precision.

Similar to convolutional layers, FCLs can also be quantized. K-means clustering can be used to aid in the compression of weights. In 2014 Gong [76] used k-means clustering on FCLs and achieved a compression ratio of more than 20× with 1% top-5 accuracy loss.

Bias terms in neural networks introduce intercepts in linear equations. They are typically regarded as constants that help the

network to train and best fit given data. Bias quantization is not widely mentioned in the literature. [120] maintained 32-bit biases while quantizing weights to 8-bit. Since biases account for minimal memory usage (e.g., 12 values for a 10-in/12-out FCL vs 120 weight values) it is recommended to leave biases in full precision. If bias quantization is performed it can be a multiplication by both the feature scale and weight scale [120], as shown in Eq. 20. However, in some circumstances they may have their own scale factor. For example, when the bit-lengths are limited to be shorter than the multiplication results.

$$s_b = s_w \times s_f, \quad \mathbf{b}_q = \mathbf{b}_r \times s_b \tag{20}$$

4.2. Quantization Methodology

We describe PTQ and QAT quantization approaches based on back-propagation use. We can also categorize them based on bit-width. In the following subsections, we introduce common quantization methods. In Section 4.2.1 low bit-width quantization is discussed. In Section 4.2.2 and 4.2.3 special cases of low bit-width quantization is discussed. In Section 4.2.5 difficulties with training quantized networks are discussed. Finally, in Section 4.2.4, alternate approached to quantization are discussed.

4.2.1. Lower Numerical Precision

Half precision floating point (16-bit floating-point, FP16) has been widely used in nVidia GPUs and ASIC accelerators with minimal accuracy loss [54]. Mixed precision training with weights, activations, and gradients using FP16 while the accumulated error for updating weights remains in FP32 has shown SOTA performance - sometimes even improved performance [172].

Researchers [165,98,233] have shown that FP32 parameters produced during training can be reduced to 8-bit integers for inference without significant loss of accuracy. Jacob [120] applied 8-bit integers for both training and inference, with an accuracy loss of 1.5% on ResNet-50. Xilinx [212] showed that 8-bit numerical precision could also achieve lossless performance with only one batch inference to adjust quantization parameters and without retraining.

Quantization can be considered an exhaustive search optimizing the scale found to reduce an error term. Given a floating-point network, the quantizer will take an initial scale, typically calculated by minimizing the l_2-error, and use it to quantize the first layer weights. Then the quantizer will adjust the scale to find the lowest output error. It performans this operation on every layer.

Integer Arithmetic-only Inference (IAI) [120] proposed a practical quantization scheme able to be adopted by industry using standard datatypes. IAI trades off accuracy and inference latency by compressing compact networks into integers. Previous techniques only compressed the weights of redundant networks resulting in better storage efficiency. IAI quantizes $z \neq 0$ in Eq. 12 requiring additional zero-point handling but resulting in higher efficiency by making use of unsigned 8-bit integers. The data-flow is described in Fig. 13. TensorFlow-Lite [120,131] deployed IAI with an accuracy loss of 2.1% using ResNet-150 on the ImageNet dataset. This is described in more detail in Section 4.3.2.

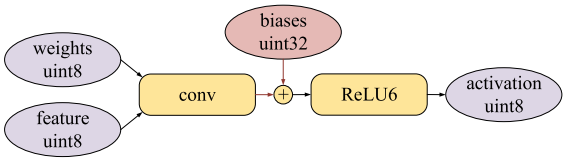


Fig. 13. Integer Arithmetic-only Inference: The convolution operation takes unsigned int8 weights and inputs, accumulates them to unsigned int32, and then performs a 32-bit addition with biases. The ReLU6 operation outputs 8-bit integers. Adopted from [120].

Datatypes other than INT8 have been used to quantize parameters. Fixed point, where the radix point is not at the right-most binary digit, is one format that has been found to be useful. It provides little loss or even higher accuracy but with a lower computation budget. Dynamic scaled fixed-point representation [233] obtained a 4× acceleration on CPUs. However, it requires specialized hardware including 16-bit fixed-point [89], 16-bit flex point [130], and 12-bit operations using dynamic fixed-point format (DFXP) [51]. The specialized hardware is mentioned in Section 4.3.3.

4.2.2. Logarithmic Quantization

Bit-shift operations are inexpensive to implement in hardware compared to multiplication operations. FPGA implementations [6] specifically benefit by converting floating-point multiplication into bit shifts. Network inference can be further optimized if weights are also constrained to be power-of-two with variable-length encoding. Logarithmic quantization takes advantage of this by being able to express a larger dynamic range compared to linear quantization.

Inspired by binarized networks [52], introduced in Section 4.2.3, Lin [156] forced the neuron output into a power-of-two value. This converts multiplications into bit-shift operations by quantizing the representations at each layer of the binarized network. Both training and inference time are thus reduced by eliminating multiplications.

Incremental Network Quantization (INQ) [269] replaces weights with power-of-two values. This reduces computation time by converting multiplies into shifts. INQ weight quantization is performed iteratively. In one iteration, weight pruning-inspired weight partitioning is performed using group-wise quantization. These weights are then fine-tuned by using a pruning-like measurement [92,88]. Group-wise retraining fine-tunes a subset of weights in full precision to preserve ensemble accuracy. The other weights are converted into power-of-two format. After multiple iterations most of the full precision weights are converted to power-of-two. The final networks have weights from 2 (ternary) to 5 bits with values near zero set to zero. Results of group-wise iterative quantization show lower error rates than a random power-of-two strategy. Specifically, INQ obtained 71× compression with 0.52% top-1 accuracy loss on the ILSVRC-2012 with AlexNet.

Logarithmic Neural Networks (LogNN) [175] quantize weights and features into a log-based representation. Logarithmic back-propagation during training is performed using shift operations. Bases other than log_2 can be used. $log_{\sqrt{2}}$ based arithmetic is described as a trade-off between dynamic range and representation precision. log_2 showed 7× compression with 6.2% top-5 accuracy loss on AlexNet, while $log_{\sqrt{2}}$ showed 1.7% top-5 accuracy loss.

Shift convolutional neural networks (ShiftCNN) [84] improve efficiency by quantizing and decomposing the real-valued weights matrix into an N times B ranged bit-shift, and encoding them with code-books $\mathbf{C}$ as shown in Eq. 21. $idx_i(n)$ is the index for the i^{th} weights in the n^{th} code-book. Each coded weight w_i can be indexed by the NB-bit expression.

$$\begin{aligned} w_i &= \sum_{n=1}^{N} \mathbf{C}_n \ [\mathrm{idx}_i(n)] \\ \mathbf{C}_n &= \left\{0, \pm 2^{-n+1}, \pm 2^{-n}, \pm 2^{-n-1}, \ldots, \pm 2^{-n-\lfloor M/2 \rfloor+2}\right\} \\ &\text{where } M = 2^B - 1 \end{aligned} \tag{21}$$

Note that the number of code-books C_n can be greater than one. This means the encoded weight might be a combination of multiple shift operations. This property allows ShiftCNN to expand to a relatively large-scale quantization or to shrink to binarized or ternary weights. We discuss ternary weights in Section 4.2.3. ShiftCNN was deployed on an FPGA platform and achieved comparable accuracy on the ImageNet dataset with 75% power saving and up to 1090× clock cycle speed-up. ShiftCNN achieves this impressive result without requiring retraining. With $N = 2$ and $B = 4$ encoding, SqueezeNet [115] has only 1.01% top-1 accuracy loss. The loss for GoogLeNet, ResNet-18, and ResNet-50 is 0.39%, 0.54%, and 0.67%, respectively, While compressing the weights into 7/32 of the original size. This implies that the weights have significant redundancy.

Based on LogNN, Cai [30] proposed improvements by disabling activation quantization to reduce overhead during inference. This also reduced the clamp bound hyperparameter tuning during training. These changes resulted in many low-valued weights that are rounded to the nearest value during encoding. As 2^n s.t. $n \in N$ increases quantized weights sparsity as n increases. In this research, n is allowed to be real-valued numbers as $n \in R$ to quantize the weights. This makes weight quantization more complex. However, a code-book helps to reduce the complexity.

In 2019, Huawei proposed DeepShift, a method of saving computing power by shift convolution [62]. DeepShift removed all floating-point multiply operations and replaced them with bit reverse and bit shift. The quantized weight W_q transformation is shown mathematically in Eq. 22, where S is a sign matrix, P is a shift matrix, and Z is the set of integers.

$$W_q = S \times 2^P, \text{ s.t. } P \in \mathbb{Z}, S \in \{-1, 0, +1\} \tag{22}$$

Results indicate that DeepShift networks cannot be easily trained from scratch. They also show that shift-format networks do not directly learn for lager datasets such as Imagenet. Similar to INQ, they show that fine-tuning a pre-trained network can improve performance. For example, with the same configuration of 32-bit activations and 6-bit shift-format weights, the top-1 ILSVRC-2012 accuracy loss on ResNet-18 for trained from scratch and tuned from a pre-trained model are 4.48% and 1.09%, respectively.

DeepShift proposes models with differential backpropagation for generating shift coefficients during the retraining process. DeepShift-Q [62] is trained with floating-point parameters in backpropagation with values rounded to a suitable format during inference. DeepShift-PS directly adopts the shift P and sign S parameters as trainable parameters.

Since logarithmic encoding has larger dynamic range, redundant networks particularly benefit. However, less redundant networks show significant accuracy loss. For example, VGG-16 which is a redundant network shows 1.31% accuracy loss on top-1 while DenseNet-121 shows 4.02% loss.

4.2.3. Plus-minus Quantization

Plus-minus quantization was in 1990 [208]. This technique reduces all weights to 1-bit representations. Similar to logarithmic quantization, expensive multiplications are removed. In this section, we provide an overview of significant binarized network results. Simons [216] and Qin [198] provide an in-depth review of BNNs.

Binarized neural networks (BNN) have only 1-bit weights and often 1-bit activations. 0 and 1 are encoded to represent −1 and +1, respectively. Convolutions can be separated into multiplies and additions. In binary arithmetic, single bit operations can be performed using *and*, *xnor*, and *bit-count*. We follow the introduction from [273] to explain bit-wise operation. Single bit fixed point dot products are calculated as in Eq. 23, where *and* is a bit-wise AND operation and *bitcount* counts the number of 1's in the bit string.

$$\boldsymbol{x} \cdot \boldsymbol{y} = \text{bitcount}(\text{and}(\boldsymbol{x}, \boldsymbol{y})), \text{s.t.} \forall i, x_i, y_i \in \{0, 1\} \tag{23}$$

This can be extended into multi-bit computations as in Eq. 24 [53]. $\boldsymbol{x}$ and $\boldsymbol{y}$ are M-bit and K-bit fixed point integers, subject to $\boldsymbol{x} = \sum_{m=0}^{M-1} c_m(\boldsymbol{x}) 2^m$ and $\boldsymbol{y} = \sum_{k=0}^{K-1} c_k(\boldsymbol{y}) 2^k$, where $(c_m(\boldsymbol{x}))_{m=0}^{M-1}$ and $(c_k(\boldsymbol{y}))_{k=0}^{K-1}$ are bit vectors.

$$\begin{aligned} &\mathrm{x} \cdot \mathrm{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \text{bitcount}[\text{and}(c_m(\mathrm{x}), c_k(\mathrm{y}))], \\ &\text{s.t. } c_m(\mathrm{x})_i, c_k(\mathrm{y})_i \in \{0, 1\} \forall i, m, k. \end{aligned} \tag{24}$$

By removing complicated floating-point multiplications, networks are dramatically simplified with simple accumulation hardware. Binarization not only reduces the network size by up-to 32×, but also drastically reduces memory usage resulting in significantly lower energy consumption [174,112]. However, reducing 32-bit parameters into a single bit results in a significant loss of information, which decreases prediction accuracy. Most quantized binary networks significantly under-perform compared to 32-bit competitors.

There are two primary methods to reduce floating-point values into a single bit: 1) stochastic and 2) deterministic [52]. Stochastic methods consider global statistics or the value of input data to determine the probability of some parameter to be −1 or +1. Deterministic binarization directly computes the bit value based on a threshold, usually 0, resulting in a sign function. Deterministic binarization is much simpler to implement in hardware.

Binary Connect (BC), proposed by Courbariaux [52], is an early stochastic approach to binarize neural networks. They binarized the weights both in forward and backward propagation. Eq. 25 shows the stochastic binarization x^b with a *hard sigmoid* probability $\sigma(x)$. Both the activations and the gradients use 32-bit single precision floating point. The trained BC network shows 1.18% classification error on the small MNIST dataset but 8.27% classification on the larger CIFAR-10 dataset.

$$\begin{aligned} &x^b = \begin{cases} +1, & \text{with probability } p = \sigma(x) \\ -1, & \text{with probability } 1 - p \end{cases} \\ &\text{where } \sigma(x) = \text{clamp}\left(\tfrac{x+1}{2}, 0, 1\right) \end{aligned} \tag{25}$$

Courbariaux extended BC networks by binarizing the activations. He named them BinaryNets [53], which is recognized as the first BNN. They also report a customized binary matrix multiplication GPU kernel that accelerates the calculation by 7×. BNN is considered the first binarized neural network where both weights and activations are quantized to binary values [216]. Considering the hardware cost of stochastic binarization, they made a trade-off to apply deterministic binarization in most circumstances. BNN reported 0.86% error on MNIST, 2.53% error on SVHN, and 10.15% error on CIFAR-10. The ILSVRC-2012 dataset accuracy results for binarized AlexNet and GoogleNet are 36.1% top-1 and 47.1%, respectively while the FP32 original networks achieve 57% and 68%, respectively [112].

Rastegari [200] explored binary weight networks (BWN) on the ILSVRC dataset with AlexNet and achieved the same classification accuracy as the single precision version. The key is a scaling factor $\alpha \in \mathbb{R}^+$ applied to an entire layer of binarized weights $\mathbf{B}$. This results in similar weights values as if they were computed using FP32 $\mathbf{W} \approx \alpha \mathbf{B}$. They also applied weight binarization on ResNet-18 and GoogLeNet, resulting in 9.5% and 5.8% top-1 accuracy loss compared to the FP32 version, respectively. They also extended binarization to activations called XNOR-Net and evaluated it on the large ILSVRC-2012 dataset. Compared to BNN, XNOR-Net also applied a scaling factor on the input feature and a rearrangement

of the network structure (swapping the convolution, activation, and BN). Finally, XNOR-Net achieved 44.2% top-1 classification accuracy on ILSVRC-2012 with AlexNet, while accelerating execution time 58× on CPUs. The attached scaling factor extended the binarized value expression, which reduced the network distortion and lead to better ImageNet accuracy.

DoReFa-Net [272] also adopts plus-minus arithmetic for quantized network. DoReFa additionally quantizes gradients to low-bit widths within 8-bit expressions during the backward pass. The gradients are quantized stochastically in back propagation. For example, it takes 1 bit to represent weights layer-wise, 2-bit activations, and 6-bits for gradients. We describe training details in Section 4.2.5. They found 9.8% top-1 accuracy loss on AlexNet with ILSVRC-2012 using the 1–2-6 combination. The result for the 1–4-32 combination is 2.9%.

Li [146] and Leng [144] showed that for ternary weights $(-1, 0, \text{and} + 1)$, in Ternary Weight Networks (TWN), only a slight accuracy loss was realized. Compared to BNN, TWN has an additional value to reduce information loss while still keeping computational complexity similar to BNN's. Ternary logic may be implemented very efficiently in hardware, as the additional value (zero) do not actually participate in computations [50]. TWN adopts the l_2-distance to find the scale and formats the weights into $-1, 0, \text{and} + 1$ with a threshold generated by an assumption that the weighs are uniformly distributed such as in $[-a, a]$. This resulted in up to 16× model compression with 3.6% ResNet-18 top-1 accuracy loss on ILSVRC-2012.

Trained Ternary Quantization (TTQ) [274] extended TWN by introducing two dynamic constraints to adjust the quantization threshold. TTQ outperformed the full precision AlexNet on the ILSVRC-2012 top-1 classification accuracy by 0.3%. It also outperformed TWN by 3%.

Ternary Neural Networks (TNN) [6] extend TWN by quantizing the activations into ternary values. A teacher network is trained with full precision and then using transfer learning the same structure is used but replacing the full precision values with a ternarized student in a layer-wise greedy method. A small difference between the real-valued teacher network and the ternarized student network is that they activate the output with a ternary output activation function to simulate the real TNN output. TNN achieves 1.67% MNIST classification error and 12.11% classification error on CIFAR10. TNN has slightly lower accuracy compared to TWN (an additional 1.02% MNIST error).

Intel proposed Fine-Grained Quantization (FGQ) [170] to generalize ternary weights by splitting them into several groups and with independent ternary values. The FGQ quantized ResNet-101 network achieved 73.85% top-1 accuracy on the ImageNet dataset (compared with 77.5% for the baseline) using four groups weights and without re-training. FGQ also showed improvements in (re) training demonstrating a top-1 accuracy improvement from 48% on non-trained to 71.1% top-1 on ResNet-50. ResNet-50's baseline accuracy is 75%. Four groups FGQ with ternary weights and low bit-width activations achieves about 9× acceleration.

MeliusNet [21] is a binary neural network that consist of two types of binary blocks. To mitigate drawbacks of low bit width networks, reduced information quality, and reduced network capacity, MeliusNet used a combination of *dense block* [22] which increases network channels by concatenating derived channels from the input to improve capacity and *improvement block* [161] which improves the quality of features by adding additional convolutional activations onto existing extra channels from *dense block*. They achieved accuracy results comparable to MobileNet on the ImageNet dataset with MeliusNet-59 reporting 70.7% top-1 accuracy while requiring only 0.532 BFLOPs. A similar sized 17 MB MobileNet required 0.569 BFLOPs achieving 70.6% accuracy.

AdderNet [35] is another technique that replaces multiply arithmetic but allows larger than 1-bit parameters. It replaces all convolutions with addition. Eq. 26 shows that for a standard convolution, AdderNet formulates it as a similarity measure problem

$$Y(m,n,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} S(\mathbf{X}(m+i,n+j,k),\mathbf{F}(i,j,k,t)) \tag{26}$$

where $\mathbf{F} \in \mathbb{R}^{d\times d\times c_{in}\times c_{\text{out}}}$ is a filter, d is the kernel size, c_{in} is an input channel and c_{out} is an output channel. $\mathbf{X} \in \mathbb{R}^{h\times w\times c_{in}}$ stands for the input feature height h and width w. With this formulation, the output Y is calculated with the similarity $S(\cdot,\cdot)$, i.e., $S(x,y) = x \times y$ for conventional convolution where the similarity measure is calculated by cross correlation. Eq. 27 mathematically describes AdderNet, which replaces the multiply with subtraction. The l_1-distance is applied to calculate the distance between the filter and the input feature. By replacing multiplications with subtractions, AdderNet speeds up inference by transforming 3.9 billion multiplications into subtractions with a loss in ResNet-50 accuracy of 1.3%.

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |\mathbf{X}(m+i,n+j,k) - \mathbf{F}(i,j,k,t)| \tag{27}$$

NAS can be applied to BNN construction. Shen [213] adopted evolutionary algorithms to find compact but accurate models achieving 69.65% top-1 accuracy on ResNet-18 with ImageNet at 2.8× speed-up. This is better performance than the 32-bit single precision baseline ResNet-18 accuracy of 69.6%. However, the search approach is time consuming taking 1440 h on an nVidia V100 GPU to search 50 k ImageNet images to process an initial network.

4.2.4. Other Approaches to Quantization

Weight sharing by vector quantization can also be considered a type of quantization. In order to compress parameters to reduce memory space usage, parameters can be clustered and shared. K-means is a widely used clustering algorithm and has been successfully applied to DNNs with minimal loss of accuracy [76,243,143] achieving 16–24 times compression with 1% accuracy loss on the ILSVRC-2012 dataset [76,243].

HashNet [37] uses a hash to cluster weights. Each hash group is replaced with a single floating-point weight value. This was applied to FCLs and shallow CNN models. They found a compression factor of 64× outperforms equivalent-sized networks on MNIST and seven other datasets they evaluated.

In 2016 Han applied Huffman coding with Deep Compression [92]. The combination of weight sharing, pruning, and huffman coding achieved 49× compression on VGG-16 with no loss of accuracy on ILSVRC-2012, which was SOTA at the time.

The Hessian method was applied to measure the importance of network parameters and therefore improve weight quantization [45]. They minimized the average Hessian weighted quantization errors to cluster parameters. They found compression ratios of 40.65 on AlexNet with 0.94% accuracy loss on ILSVRC-2012. Weight regularization can slightly improve the accuracy of quantized networks by penalizing weights with large magnitudes [215]. Experiments showed that l_2 regularization improved 8-bit quantized MobileNet top-1 accuracy by 0.23% on ILSVRC-2012.

BN has proved to have many advantages including addressing the internal covariate shift issue [119]. It can also be considered a type of quantization. However, quantization performed with BN may have numerical instabilities. The BN layer has nonlinear square and square root operations. Low bit representations may be problematic when using non-linear operations. To solve this, l_1-norm BN [245] has only linear operations in both forward and backward training. It provided 1.5× speedup at half the power

on FPGA platforms and can be used with both training and inference.

4.2.5. Quantization-aware Training

Most quantization methods use a global (layer-wise) quantization to reduce the full precision model into a reduced bit model. Thus can result in non-negligible accuracy loss. A significant drawback of quantization is information loss caused by the irreversible precision reducing transform. Accuracy loss is particularly visible in binary networks and shallow networks. Applying binary weights and activations to ResNet-34 or GoogLeNet resulted in 29.10% and 24.20% accuracy loss, respectively [53]. It has been shown that backward propagation fine-tunes (retrains) a quantized network and can recover losses in accuracy caused by the quantization process [171]. The retraining is even resilient to binarization information distortions. Thus training algorithms play a crucial role when using quantization. In this section, we introduce (re) training of quantized networks. BNN Training: For a binarized network that has binary valued weights it is not effective to update the weights using gradient decent methods due to typically small derivatives. Early quantized networks were trained with a variation of Bayesian inference named Expectation Back Propagation (EBP) [220,41]. This method assigns limited parameter precision (e.g., binarized) weights and activations. EBP infers networks with quantized weights by updating the posterior distributions over the weights. The posterior distributions are updated by differentiating the parameters of the backpropagation.

BinaryConnect [52] adopted the probabilistic idea of EBP but instead of optimizing the weights posterior distribution, BC preserved floating-point weights for updates and then quantized them into binary values. The real-valued [4] weights update using the back propagated error by simply ignoring the binarization in the update.

A binarized Network has only 1-bit parameters - ± 1 quantized from a sign function. Single bit parameters are non-differentiable and therefore it is not possible to calculate gradients needed for parameter updating [208]. SGD algorithms have been shown to need 6 to 8 bits to be effective [180]. To work around these limitations the Straight-Through Estimator (STE), previously introduced by Hinton [102], was applied for propagating gradients by using discretization [112]. Eq. 28 show the STE for sign binarization, where c denotes the cost function, w_r is the real-valued weights, and w_b is the binarized weight produced by the *sign* function. STE bypasses the binarization function to directly calculate real-valued gradients. The floating-point weights are then updated using methods like SGD. To avoid real-valued weights approaching infinity, BNNs typically clamp floating-point weights to the desired range of ± 1 [112].

$$\begin{aligned} &\text{Forward}: \quad w_b = \text{sign}(w_r) \\ &\text{Backward}: \quad \tfrac{\partial c}{\partial w_r} = \tfrac{\partial c}{\partial w_b}\mathbf{1}_{|w_r|\leqslant 1} \end{aligned} \tag{28}$$

Unlike the forward phase where weights and activations are produced with deterministic quantization, in the gradient phase, the low bit gradients should be generated by stochastic quantization [89,271]. DoReFa [272] first successfully trained a network with gradient bit-widths less than eight and achieved a comparable result with k-bit quantization arithmetic. This low bit-width gradient scheme could accelerate training in edge devices with little impact to network accuracy but minimal inference acceleration compared to BNNs. DoReFa quantizes the weights, features, and gradients into many levels obtaining a larger dynamic range than BNNs. They trained AlexNet on ImageNet from scratch with 1-bit weights, 2-bit activations, and 6-bit gradients. They obtained 46.1% top-1 accuracy (9.8% loss comparing with the full precision counterpart). Eq. 29 shows the weight quantizing approach. w is the weights (the same as in Eq. 28), limit is a limit function applied to the weights keeping them in the range of [0, 1], and quantize_k quantizes the weights into k-levels. Feature quantization is performed using the $f_\alpha^k = \text{quantize}_k$ function.

$$\begin{aligned} & f_w^k = 2\text{quantize}_k(\text{limit}(w_r)) - 1 \\ \text{where} \quad & \text{quantize}_k(w_r) = \tfrac{1}{2^k-1}\text{round}\Big(\big(2^k-1\big)w_r\Big), \\ \text{and} \quad & \text{limit}(x) = \tfrac{\tanh(x)}{2\max(|\tanh(x)|)} + \tfrac{1}{2} \end{aligned} \tag{29}$$

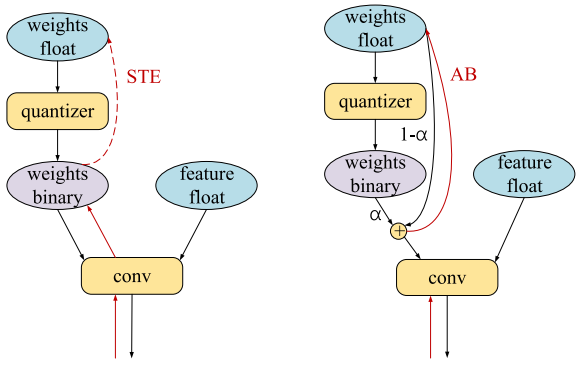
In DoReFa, gradient quantization is shown in Eq. 30, where $\text{d}r = \partial c/\partial r$ is the backprogagated gradient of the cost function c to output r.

$$\tilde{f}_\gamma^k = 2\max_0(|\text{d}r|)\left[\text{quantize}_k\left(\frac{\text{d}r}{2\max_0(|\text{d}r|)} + \frac{1}{2}\right) - \frac{1}{2}\right] \tag{30}$$

As in deep feed forward networks, the exploding gradient problem can cause BNN's not to train. To address this issue, Hou [104] formulated the binarization effect on the network loss as an optimization problem which was solved by a proximal Newton's algorithm with diagonal Hessian approximation that directly minimizes the loss with respect to the binary weights. This optimization found 0.09% improvement on MNIST dataset compared with BNN.

Alpha-Blending (AB) [162] was proposed as a replacement for STE. Since STE directly sets the quantization function gradients to 1, a hypothesis was made that STE tuned networks could suffer accuracy losses. Fig. 14 shows that AB introduces an additional scale coefficient α. Real-valued weights and quantized weights are both kept. During training α is gradually raised to 1 until a fully quantized network is realized.

Low Numerical Precision Training: Training with low numerical precision involves taking the low precision values into both forward and backward propagation while maintaining the full precision accumulated results. Mixed Precision [172,54] training uses FP16 or 16-bit integer (INT16) for weight precision. This has been shown to be inaccurate for gradient values. As shown in Fig. 15, full precision weights are maintained for gradient updating, while other operands use half-float. A *loss scaling* technique is applied to keep very small magnitude gradients from affecting the computation since any value less than 2^{-24} becomes zero in half-precision [172]. Specifically, a scaler is introduced to the loss value before backpropagation. Typically, the scaler is a bit-shift optimal value 2^n obtained empirically or by statistical information.



(a) Straight-through Estimator (b) Alpha-Blending Approach

Fig. 14. STE and AB: STE directly bypasses the quantizer while AB calculates gradients for real-valued weights by introducing additional coefficients α [162].

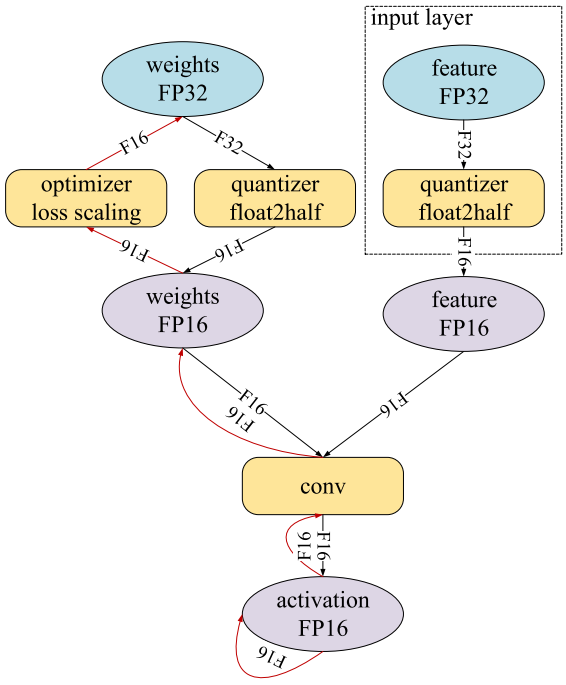


Fig. 15. Mixed Precision Training [172]: FP16 is applied in the forward and backward pass, while FP32 weights are maintained for the update.

In TensorFlow-Lite [120], training proceeds with real values while quantization effects are simulated in the forward pass. Real-valued parameters are quantized to lower precision before convolutional layers. BN layers are folded into convolution layers. More details are described in Section 4.3.2.

As in binarized networks, STE can also be applied to reduced precision training such as 8-bit integers [131].

4.3. Quantization Deployment

In this section, we describe implementations of quantization deployed in popular frameworks and hardware. In Section 4.3.1 we give an introduction to deployment issues. In Section 4.3.2, we discuss deep learning libraries and frameworks. We introduce their specification in Table 2 and then compare their performance in Table 3. We also discuss hardware implementations of DNNs in Section 4.3.3. Dedicated hardware is designed or programmed to support efficient processing of quantized networks. Specialized CPU and GPU operations are discussed. Finally, in Section 4.3.4 we discuss DNN compilers.

4.3.1. Deployment Introduction

With significant resource capability, large organizations and institutions usually have their own proprietary solutions for applications and heterogeneous platforms. Their support to the quantization is either inference only or as well as training. The frameworks don't always follow the same idea of quantization. Therefore there are differences between them, so performs.

With DNNs being applied in many application areas, the issue of efficient use of hardware has received considerable attention. Multicore processors and accelerators have been developed to accelerate DNN processing. Many types of accelerators have been deployed, including CPUs with instruction enhancements, GPUs, FPGAs, and specialized AI accelerators. Often accelerators are incorporated as part of a heterogeneous system. A Heterogeneous System Architecture (HSA) allows the different processors to integrate into a system to simultaneously access shared memory. For example, CPUs and GPUs using cache coherent shared virtual memory on the same System of Chip (SoC) or connected by PCIe with platform atomics can share the same address space [74]. Floating-point arithmetic units consume more energy and take longer to compute compared to integer arithmetic units. Consequently, low-bitwidth architectures are designed to accelerate computation [179]. Specialized algorithms and efficient hardware can accelerate neural network processing during both training and inference [202].

4.3.2. Efficient Kernels

Typically low precision inference in only executed on convolutional layers. Intermediate values passed between layers use 32-bit floating-point. This makes many of the frameworks amenable to modifications.

Table 2 gives a list of major low precision acceleration frameworks and libraries. Most of them use INT8 precision. We will next describe some popular and open-source libraries in more detail.

Tensor RT [232,242] is an nVidia developed C++ library that facilitates high-performance inference on NVIDIA GPUs. It is a low precision inference library that eliminates the bias term in convolutional layers. It requires a calibration set to adjust the quantization thresholds for each layer or channel. Afterwards the quantized parameters are represented by 32-bit floating-point scalar and INT8 weights.

Tensor RT takes a pre-trained floating-point model and generates a reusable optimized 8-bit integer or 16-bit half float model. The optimizer performs network profiling, layer fusion, memory management, and operation concurrency. Eq. 31 shows the convolution-dequantization dataflow in Tensor RT for 8-bit integers. The intermediate result of convolution by INT8 input feature

Table 2
Low Precision Libraries Using Quantization: QAT is quantization-aware training, PTQ is post-training quantization, and offset indicates the zero point z in Eq. 12.

Name	Institution	Core Lib	Precision	Method	Platform	Open-sourced
ARM CMSIS NN [129]	Arm	CMSIS	8-bit	deploy only	Arm Cortex-M Processor	No
MACE [247]	XiaoMi	-	8-bit	QAT and PTQ	Mobile - CPU, Hexagon Chips, MTK APU	Yes
MKL-DNN [204]	Intel	-	8-bit	PTQ, mixed offset, and QAT	Intel AVX Core	Yes
NCNN [229]	Tencent	-	8-bit	PTQ w/o offset	Mobile Platform	Yes
Paddle [13]	Baidu	-	8-bit	QAT and PTQ w/o offset	Mobile Platform	Yes
QNNPACK [61]	Fackbook	-	8-bit	PTQ w/ offset	Mobile Platform	Yes
Ristretto [90]	LEPS	gemm	3 method	QAT	Desktop Platform	Yes
SNPE [228]	Qualcomm	-	16/8-bit	PTQ w/ offset, max–min	Snapdragon CPU, GPU, DSP	No
Tensor-RT [173]	nVidia	-	8-bit	PTQ w/o offset	nVidia GPU	Yes
TF-Lite [1]	Google	gemmlowp	8-bit	PTQ w/ offset	Mobile Platform	Yes

Table 3
Low Precision Libraries versus Accuracy for Common Networks in Multiple Frameworks.

Name	Framework	Method	Accuracy Float		Accuracy Quant		Accuracy Diff	
			Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
AlexNet	TensorRT [173]	PTQ, w/o offset	57.08%	80.06%	57.05%	80.06%	−0.03%	0.00%
	Ristretto [90]	Dynamic FP	56.90%	80.09%	56.14%	79.50%	−0.76%	−0.59%
	Ristretto [90]	Minifloat	56.90%	80.09%	52.26%	78.23%	−4.64%	−1.86%
	Ristretto [90]	Pow-of-two	56.90%	80.09%	53.57%	78.25%	−3.33%	−1.84%
GoogleNet	NCNN [28]	PTQ, w/o offset	68.50%	88.84%	68.62%	88.68%	0.12%	−0.16%
	TensorRT [173]	PTQ, w/o offset	68.57%	88.83%	68.12%	88.64%	−0.45%	−0.19%
	Ristretto [90]	Dynamic FP	68.93%	89.16%	68.37%	88.63%	−0.56%	−0.53%
	Ristretto [90]	Minifloat	68.93%	89.16%	64.02%	87.69%	−4.91%	−1.47%
	Ristretto [90]	Pow-of-two	68.93%	89.16%	57.63%	81.38%	−11.30%	−7.78%
Inception v3	TF-Lite [77]	PTQ	78.00%	93.80%	77.20%	-	−0.80%	-
	TF-Lite [77]	QAT	78.00%	93.80%	77.50%	93.70%	−0.50%	−0.10%
MobileNet v1	NCNN [28]	PTQ, w/o offset	67.26%	87.92%	66.74%	87.43%	−0.52%	−0.49%
	Paddle [13]	QAT + Pruning	70.91%	-	69.20%	-	−1.71%	-
	TF-Lite [77]	PTQ	70.90%	-	65.70%	-	−5.20%	-
	TF-Lite [77]	QAT	70.90%	-	70.00%	-	−0.90%	-
MobileNet v2	QNNPACK [61]	PTQ, w/ offset	71.90%	-	72.14%	-	0.24%	-
	TF-Lite [77]	PTQ	71.90%	-	63.70%	-	−8.20%	-
	TF-Lite [77]	QAT	71.90%	-	70.90%	-	−1.00%	-
ResNet-101	TensorRT [173]	PTQ, w/o offset	74.39%	91.78%	74.40%	91.73%	0.01%	−0.05%
	TF-Lite [77]	PTQ	77.00%	-	76.80%	-	−0.20%	-
ResNet-152	TensorRT [173]	PTQ, w/o offset	74.78%	91.82%	74.70%	91.78%	−0.08%	−0.04%
ResNet-18	NCNN [28]	PTQ, w/o offset	65.49%	86.56%	65.30%	86.52%	−0.19%	−0.04%
ResNet-50	NCNN [28]	PTQ, w/o offset	71.80%	89.90%	71.76%	90.06%	−0.04%	0.16%
	TensorRT [173]	PTQ, w/o offset	73.23%	91.18%	73.10%	91.06%	−0.13%	−0.12%
SqueezeNet	NCNN [28]	PTQ, w/o offset	57.78%	79.88%	57.82%	79.84%	0.04%	−0.04%
	Ristretto [90]	Dynamic FP	57.68%	80.37%	57.21%	79.99%	−0.47%	−0.38%
	Ristretto [90]	Minifloat	57.68%	80.37%	54.80%	78.28%	−2.88%	−2.09%
	Ristretto [90]	Pow-of-two	57.68%	80.37%	41.60%	67.37%	−16.08%	−13.00%
VGG-19	TensorRT [173]	PTQ, w/o offset	68.41%	88.78%	68.38%	88.70%	−0.03%	−0.08%

$\mathbf{F}_{i8}$ and weights $\mathbf{W}_{i8}$ are accumulated into INT32 tensor $\mathbf{O}_{i32}$. They are dequantized by dividing by the feature and weight scales $s_f,\ s_w$.

$$\mathbf{O}_{i32} = \mathbf{F}_{i8} * \mathbf{W}_{i8}, \quad \mathbf{O}_{f32} = \frac{\mathbf{O}_{i32}}{s_f \times s_w} \tag{31}$$

Tensor RT applies a variant of *max-abs* quantization to reduce storage requirements and calculation time of the zero point term z in Eq. 15 by finding the proper threshold instead of the absolute value in the floating-point tensor. KL-divergence is introduced to make a trade-off between numerical dynamic range and precision of the INT8 representation [173]. KL calibration can significantly help to avoid accuracy loss.

The method traverses a predefined possible range of scales and calculates the KL-divergences for all the points. It then selects the scale which minimizes the KL-divergence. KL-divergence is widely used in many post training acceleration frameworks. nVidia found a model calibrated with 125 images showed only 0.36% top-1 accuracy loss using GoogLeNet on the Imagenet dataset.

Intel MKL-DNN [204] is an optimized computing library for Intel processors with Intel AVX-512, AVX-2, and SSE4.2 Instruction Set Architectures (ISA). The library uses FP32 for training and inference. Inference can also be performed using 8-bits in convolutional layers, ReLU activations, and pooling layers. It also uses Winograd convolutions. MKL-DNN uses *max-abs* quantization shown in Eq. 15, where the feature adopts unsigned 8-bit integer $n_f = 256$ and signed 8-bit integer weights $n_w = 128$. The rounding function $f(\cdot)$ in Eq. 12 uses nearest integer rounding. Eq. 32 shows the quantization applied on a given tensor or each channel in a tensor. The maximum of weights R_w and features R_f is calculated from the maximum of the absolute value (nearest integer rounding) of the tensor $\mathbb{T}_f$ and $\mathbb{T}_w$. The feature scale s_f and weights scale s_w are generated using R_w and R_f. Then quantized 8-bit signed integer weights $\mathbf{W}_{s8}$, 8-bit unsigned integer feature $\mathbf{F}_{u8}$ and 32-bit unsigned integer biases $\mathbf{B}_{u32}$ are generated using the scales and a nearest rounding function $\|\cdot\|$.

$$\begin{aligned} R_{\{f,w\}} &= max\left((abs(\mathbb{T}_{\{f,w\}}))s_f = \frac{255}{R_f},\ s_w = \frac{127}{R_w}\mathbf{W}_{s8} \right. \\ &= \|s_w \times \mathbf{W}_{f32}\| \in [-127, 127]\mathbf{F}_{u8} = \|s_f \times \mathbf{F}_{f32}\| \\ &\left. \in [0, 255]\mathbf{B}_{s32} = \|s_f \times s_w \times \mathbf{B}_{f32}\| \in \left[-2^{31}, 2^{31}-1\right] \right) \end{aligned} \tag{32}$$

An affine transformation using 8-bit multipliers and 32-bit accumulates results in Eq. 33 with the same scale factors as defined in Eq. 32 and $*$ denoting convolution. It is an approximation since rounding is ignored.

$$\begin{aligned} \mathbf{O}_{s32} &= \mathbf{W}_{s8} * \mathbf{F}_{u8} + \mathbf{b}_{s32} \\ &\approx s_f s_w (\mathbf{W}_{f32} * \mathbf{F}_{f32} + \mathbf{b}_{f32}) \\ &= s_f \times s_w \times \mathbf{O}_{f32} \end{aligned} \tag{33}$$

Eq. 34 is the affine transformation with FP32 format. D is the dequantization factor.

$$\begin{aligned} \mathbf{O}_{f32} &= \mathbf{W}_{f32} * \mathbf{F}_{f32} + \mathbf{b}_{f32} \\ &\approx \tfrac{1}{s_f s_w}\mathbf{O}_{s32} = D \times \mathbf{O}_{s32} \\ &\text{where } D = \tfrac{1}{s_f s_w} \end{aligned} \tag{34}$$

Weight quantization is done prior to inference. Activation quantization factors are prepared by sampling the validation dataset to find a suitable range (similar to Tensor RT). The quantization factors can be either FP32 in the supported devices, or rounded to the nearest power-of-two format to enable bit-shifts. Rounding reduces accuracy by about 1%.

MKL-DNN assumes activations are non-negative (ReLU activated). Local Response Normalization (LRN), a function to pick

the local maximum in a local distribution, is used to avoid overfitting. BN, FCL, and soft-max using 8-bit inference are not currently supported.

TensorFlow-Lite (TF-Lite) [1] is an open source framework by Google for performing inference on mobile or embedded devices. It consists of two sets of tools for converting and interpreting quantized networks. Both PTQ and QAT are available in TF-Lite.

GEMM low-precision (Gemmlowp) [78] is a Google open source gemm library for low precision calculations on mobile and embedded devices. It is used in TF-Lite. Gemmlowp uses asymmetric quantzation as shown in Eq. 35 where $\mathbf{F}, \mathbf{W}, \mathbf{O}$ denotes feature, weights and output, respectively. s_f, s_w are the scales for feature and weights, respectively. $\mathbf{F}_{f32}$ is Feature value in 32-bit floating. Similarly, $\mathbf{W}_{f32}$ is the Weight value in 32-bit floating point. $\mathbf{F}_q, \mathbf{W}_q$ are the quantized Features and Weights, respectively. Asymmetric quantization introduces the zero points ($\mathbf{z}_f$ and $\mathbf{z}_w$). This produces a more accurate numerical encoding.

$$\begin{aligned}\mathbf{O}_{f32} &= \mathbf{F}_{f32} * \mathbf{W}_{f32} \\ &= s_f \times (\mathbf{F}_q + \mathbf{z}_f) * s_w \times (\mathbf{W}_q + \mathbf{z}_w) \\ &= s_f \times s_w \times \underline{(\mathbf{F}_q + \mathbf{z}_f) * (\mathbf{W}_q + \mathbf{z}_k)}\end{aligned} \tag{35}$$

The underlined part in Eq. 35 is the most computationally intensive. In addition to the convolution, the zero point also requires calculation. Gemmlowp reduces many multi-add operations by multiplying an all-ones matrix as the *bias* matrix P and Q in Eq. 36. This allows four multiplies to be dispatched in a three stage pipeline [131], to produce the quantized output $\mathbf{O}_q$. $\mathbf{F}, \mathbf{W}, \mathbf{z}$ are the same as in Eq. 35.

$$\begin{aligned}\mathbf{O}_q &= (\mathbf{F}_q + \mathbf{z}_f \times P) * (\mathbf{W}_q + \mathbf{z}_w \times Q) \\ &= \mathbf{F}_q * \mathbf{W}_q + \mathbf{z}_f \times \mathbf{P} \times \mathbf{W}_q + \mathbf{z}_w \times \mathbf{Q} \times \mathbf{F}_q + \mathbf{z}_f \times \mathbf{z}_w \times \mathbf{P} \times \mathbf{Q}\end{aligned} \tag{36}$$

Ristretto [90] is a tool for Caffe quantization. It uses retraining to adjust the quantized parameters. Ristretto uses a three-part quantization strategy: 1) a modified fixed-point format Dynamic Fixed Point (DFP) which permits the limited bit-width precision to dynamically carry data, 2) bit-width reduced floating-point numbers called mini float which follows the IEEE-754 standard [219], and 3) integer power of 2 weights that force parameters into power of 2 values to replace multiplies with bit shift operations.

DPF is shown in Eq. 37 where s takes one sign bit, FL denotes the fractional length, and x is the mantissa. The total bit-width is B. This quantization can encode data from various ranges to a proper format by adjusting the fractional length.

$$(-1)^s \cdot 2^{-\mathrm{FL}} \sum_{i=0}^{B-2} 2^i \cdot x_i \tag{37}$$

A bit shift convolution conversion is shown in Eq. 38. The convolution by input $\mathbf{F}_j$ and weights $\mathbf{W}_j$ and bias $\mathbf{b}_i$ are transformed into shift arithmetic by rounding the weights to the nearest power of 2 values. Power of 2 weights provides inference acceleration while dynamic fixed point provides better accuracy.

$$\begin{aligned}\mathbf{O}_i &= \sum_j [\mathbf{F}_j \cdot \mathbf{W}_j] + \mathbf{b}_i \\ &\approx \sum_j [\mathbf{F}_j \ll \mathrm{round}(\log_2(\mathbf{W}_j))] + \mathbf{b}_i\end{aligned} \tag{38}$$

NCNN [229] is a standalone framework from Tencent for efficient inference on mobile devices. Inspired by Ristretto and Tensor-RT, it works with multiple operating systems and supports low precision inference [28]. It performs channel-wise quantization with KL calibration. The quantization results in 0.04% top-1 accuracy loss on ILSVRC-2012. NCNN has implementations optimized for ARM NEON. NCNN also replaces 3×3 convolutions with simpler Winograd convolutions [135].

Mobile AI Compute Engine (MACE) [247] from Xiaomi supports both post-training quantization and quantization-aware training. Quantization-aware training is recommended as it exhibits lower accuracy loss. Post-training quantization requires statistical information from activations collected while performing inference. This is typically performed with batch calibration of input data. MACE also supports processor implementations optimized for ARM NEON and Qualcomm's Hexagon digital signal processor. OpenCL acceleration is also supported. Winograd convolutions can be applied for further acceleration as discussed in Section 4.2.2.

Quantized Neural Network PACKage (QNNPACK) [61] is a Facebook produced open-source library optimized for edge computing especially for mobile low precision neural network inference. It has the same method of quantization as TF-Lite including using a zero-point. The library has been integrated into PyTorch [193] to provide users a high-level interface. In addition to Winograd and FFT convolution operations, the library has optimized gemm for cache indexing and feature packing. QNNPACK has a full compiled solution for many mobile devices and has been deployed on millions of devices with Facebook applications.

Panel Dot product (PDOT) is a key feature of QNNPACK's highly efficient gemm library. It assumes computing efficiency is limited with memory, cache, and bandwidth instead of Multiply and Accumulate (MAC) performance. PDOT computes multiple dot products in parallel as shown in Fig. 16. Rather than loading just two operands per MAC operation, PDOT loads multiple columns and rows. This improves convolution performance about 1.41× 2.23× speedup for MobileNet on mobile devices [61].

Paddle [13] applies both QAT and PTQ quantization with using zero-points. The dequantization operation can be performed prior to convolution as shown in Eq. 39. Paddle uses this feature to do floating-point gemm-based convolutions with quantize-dequantized weights and features within the framework datapath. It introduces quantization error while maintaining the data in format of floating-point. This quantize-dequantize-convolution pipeline is called simu-quantize and its results are approximately equal to a FP32->INT8->Convolutional->FP32 (quantize - convolutional - dequantize) three stage model.

Simu-quantize maintains the data at each phase in 32-bit floating-point facilitating backward propagation. In the Paddle framework, during backpropagation, gradients are added to the original 32-bit floating-point weights rather than the quantized or the quantize-dequantized weights.

$$\mathbf{O}_{f32} = \left(\frac{\mathbf{F}_q}{(n-1)} \times \mathbf{F}_{max}\right) * \left(\frac{\mathbf{W}_q}{(n-1)} \times \mathbf{W}_{max}\right) \tag{39}$$

Paddle uses *max-abs* in three ways to quantize parameters: 1) the average of the max absolute value in a calculation window, 2) the max absolute value during a calculation window, and 3) a sliding

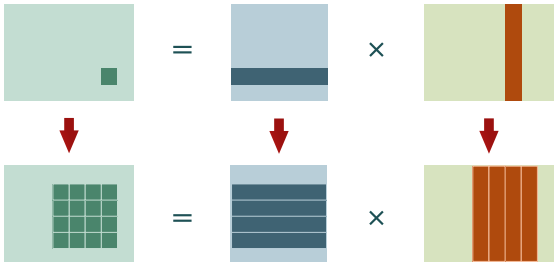


Fig. 16. PDOT: computing dot product for several points in parallel.

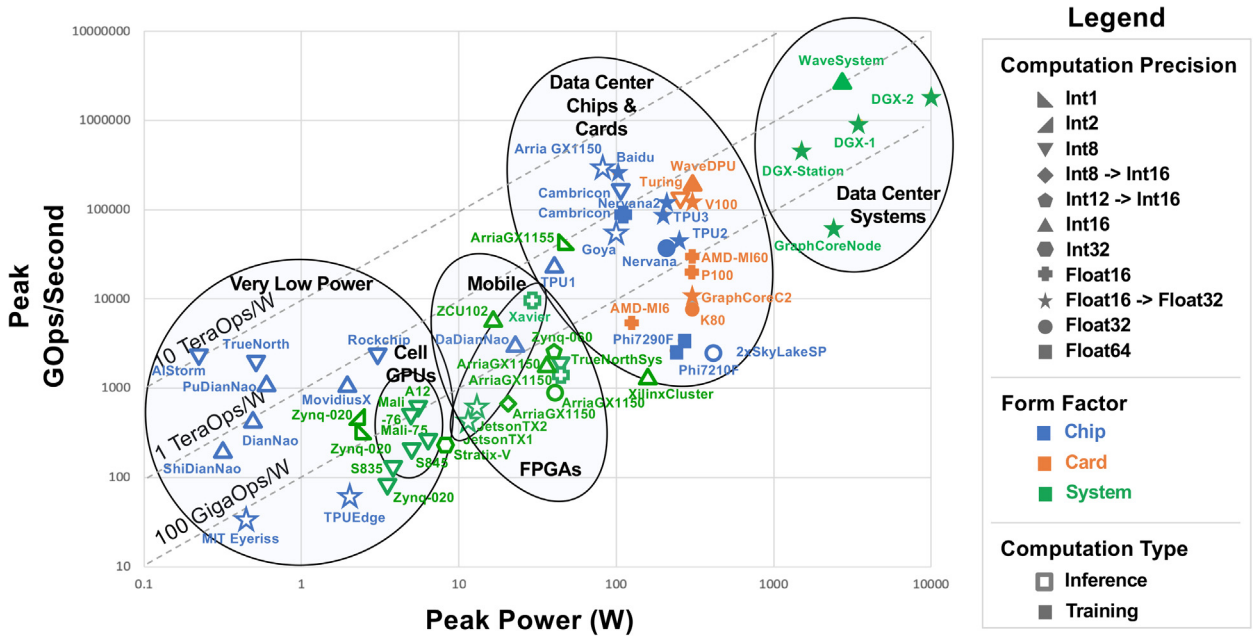


Fig. 17. Hardware platforms for neural networks efficiency deploy, adopted from [202].

average of the max absolute value of the window. The third method is described in Eq. 40, where V is the max absolute value in the current batch, V_t is the average value of the sliding window, and k is a coefficient chosen by default as 0.9.

The Paddle framework uses a specialized toolset, PaddleSlim, which supports Quantization, Pruning, Network Architecture Search, and Knowledge Distilling. They found 86.47% size reduction of ResNet-50, with 1.71% ILSVRC-2012 top-1 accuracy loss.

$$V_t = (1-k) \times V + k \times V_{t-1} \tag{40}$$

4.3.3. Hardware Platforms

Fig. 17 shows AI chips, cards, and systems plotted by peak operations verses power in log scale originally published in [202]. Three normalizing lines are shown at 100 GOPS/Watt, 1 TOP/Watt, and 10 TOPs/Watt. Hardware platforms are classified along several dimensions including: 1) training or inference, 2) chip, card, or system form factors, 3) datacenter or mobile, and 4) numerical precision. We focus on low precision general and specialized hardware in this section. Programmable Hardware:

Quantized networks with less than 8-bits of precision are typically implemented in FPGAs but may also be executed on general purpose processors.

BNN's have been implemented on a Xilinx Zynq heterogeneous FPGA platform [267]. They have also been implemented on Intel Xeon CPUs and Intel Arria 10 FPGA heterogeneous platforms by dispatching bit operation to FPGAs and other operations to CPUs [178]. The heterogeneous system shares the same memory address space. Training is typically mapped to CPUs. FINN [231] is a specialized framework for BNN inference on FPGAs. It contains binarized fully connected, convolutional, and pooling layers. When deployed on a Zynq-7000 SoC, FINN has achieved 12.36 million images per second on the MNIST dataset with 4.17% accuracy loss.

Binarized weights with 3-bit features have been implemented on Xilinx Zynq FPGAs and Arm NEON processors [196]. The first and last layer of the network use 8-bit quantities but all other layers use binary weights and 3-bit activation values. On an embedded platform, Zynq XCZU3EG, they performed 16 images per second for inference. To accelerate Tiny-YOLO inference, significant efforts were taken including: 1) replacing max-pool with stride 2 convolution, 2) replacing leaky ReLU with ReLU, and 3) revising the hidden layer output channel. The improved efficiency on the FPGA from 2.5 to 5 frames per second with 1.3% accuracy loss.

TNN [6] is deployed on an FPGA with specialized computation units optimized for ternary value multiplication. A specific FPGA structure (dimensions) is determined during synthesis to improve hardware efficiency. On the Sakura-X FPGA board they achieved 255 k MNIST image classifications per second with an accuracy of 98.14%. A scalable design implemented on a Xilinx Virtex-7 VC709 board dramatically reduced hardware resources and power consumption but at a significantly reduced throughput of 27 k CIFAR-10 images per second [197]. Power consumption for CIFAR-10 was 6.8 Watts.

Reducing hardware costs is a key objective of logarithmic hardware. Xu [249] adopted $\sqrt{2}$ based logarithmic quantization with 5-bits of resolution. This showed 50.8% top-1 accuracy and dissipated a quarter of the power while using half the chip area. Half precision inference has a top-1 accuracy of 53.8%.

General Hardware: In addition to specialized hardware, INT8 quantization has been widely adopted in many general purpose processor architectures. In this section we provide a high-level overview. A detailed survey on hardware efficiency for processing DNNs can be found in [202].

CNN acceleration on ARM CPUs was originally implemented by ARM advanced SIMD extensions known as NEON. The ARM 8.2 ISA extension added NEON support for 8-bit integer matrix operations [8]. These were implemented in the CPU IP cores Cortex-A75 and A55 [9] as well as the Mali-G76 GPU IP core [10]. These cores have been integrated into the Kirin SoC by Huawei, Qualcomm Snapdragon SoC, MediaTek Helio SoC, and Samsung Exynos [116]. For example on Exynos 9825 Octa, 8-bit integer quantized MobileNet v2 can process an image in 19 ms (52 images per second) using the Mali-G76 [116].

Intel improved the integer performance about 33% with Intel Advanced Vector Extension 512 (AVX-512) ISA [204]. This 512-bit SIMD ISA extension included a Fused Multiply–Add (FMA) instruction.

Low precision computation on nVidia GPUs was enabled since the Pascal series of GPUs [184]. The Turing GPU architecture [188] introduced specialized units to processes INT4 and INT8. This

provides real-time integer performance on AI algorithms used in games. For embedded platforms, nVidia developed Jetson platforms [187]. They use CUDA Maxwell cores [183] that can process half-precision types. For the data center, nVidia developed the extremely high performance DGX system [185]. It contains multiple high-end GPUs interconnected using nVidia's proprietary bus nVLINK. A DGX system can perform 4-bit integer to 32-bit floating point operations.

4.3.4. DNN Compilers

Heterogeneous neural networks hardware accelerators are accelerating deep learning algorithm deployment [202]. Often exchange formats can be used to import/export models. Further, compilers have been developed to optimize models and generate code for specific processors. However several challenges remain:

- Network Parsing: Developers design neural network models on different platforms using various frameworks and programming languages. However, they have common parts, such as convolution, activation, pooling, etc. Parsing tools analyze the model compositions and transfer them into the unified representation.
- Structure Optimization: The model may contain operations used in training that aren't required for inference. Tool-kits and compilers should optimize these structures (e.g., BN folding as discussed in Section 2.5).
- Intermediate Representation (IR): An optimized model should be properly stored for further deployment. Since the inference engine is uncertain, the stored IR should include the model architecture and the trained weights. A compiler can then read the model and optimize it for a specific inference engine.
- Compression: Compilers and optimizers should optionally be able to automatically compress arbitrary network structures using pruning and quantization.
- Deployment: The final optimized model should be mapped [4] to the target engine(s) which may be heterogeneous.Open Neural Network Exchange (ONNX) [190] is an open-source tool to parse AI models written for a variety diverse frameworks. It imports and exports models using an open-source format facilitating the translation of neural network models between frameworks. It is thus capable of network parsing provided low-level operations are defined in all target frameworks.

TVM [36], Glow [205], OpenVINO [118], and MLIR [134] are deep learning compilers. They differ from frameworks such as Caffe in that they store intermediate representations and optimize those to map models onto specific hardware engines. They typically integrate both quantization-aware training and calibration-based post-training quantization. We summarize key features below. They perform all the operations noted in our list. A detailed survey can be found in [149].

TVM [36] leverages the efficiency of quantization by enabling deployment of quantized models from PyTorch and TF-Lite. As a compiler, TVM has the ability to map the model on general hardware such as Intel's AVX and nVidia's CUDA.

Glow [205] enables quantization with zero points and converts the data into 8-bit signed integers using a calibration-based method. Neither Glow or TVM currently support quantization-aware training although they both announced future support for it [205].

MLIR [134] and OpenVINO [118] have sophisticated quantization support including quantization-aware training. OpenVINO integrates it in TensorFlow and PyTorch while MLIR natively supports quantization-aware training. This allows users to fine-tune an optimized model when it doesn't satisfy accuracy criteria.

4.4. Quantization Reduces Over-fitting

In addition to accelerating neural networks, quantization has also been found in some cases to result in higher accuracy. As examples: 1) 3-bit weights VGG-16 outperforms its full precision counterpart by 1.1% top-1 [144], 2) AlexNet reduces 1.0% top-1 error of the reference with 2-bit weights and 8-bit activations [66], 3) ResNet-34 with 4-bit weights and activation obtained 74.52% top-1 accuracy while the 32-bit version is 73.59% [174], 4) Zhou showed a quantized model reduced the classification error by 0.15%, 2.28%, 0.13%, 0.71%, and 1.59% on AlexNet, VGG-16, GoogLeNet, ResNet-18 and ResNet-50, respectively [269], and 5) Xu showed reduced bit quantized networks help to reduce over-fitting on Fully Connected Networks (FCNs). By taking advantage of strict constraints in biomedical image segmentation they improved segmentation accuracy by 1% combined with a 6.4× memory usage reduction [251].

5. Summary

In this section we summarize the results of Pruning and Quantization.

5.1. Pruning

Section 3 shows pruning is an important technique for compressing neural networks. In this paper, we discussed pruning techniques categorized as 1) static pruning and 2) dynamic pruning. Previously, static pruning was the dominant area of research. Recently, dynamic pruning has become a focus because it can further improve performance even if static pruning has first been performed.

Pruning can be performed in multiple ways. Element-wise pruning improves weight compression and storage. [4] Channel-wise and shape-wise pruning can be accelerated with specialized hardware and computation libraries. Filter-wise and layer-wise pruning can dramatically reduce computational complexity.

Though pruning sometimes introduces incremental improvement in accuracy by escaping a local minima [12], accuracy improvements are better realized by switching to a better network architecture [24]. For example, a separable block may provide better accuracy with reduced computational complexity [105]. Considering the evolution of network structures, performance may also be bottlenecked by the structure itself. From this point of view, Network Architecture Search and Knowledge Distillation can be options for further compression. Network pruning can be viewed as a subset of NAS but with a smaller searching space. This is especially true when the pruned architecture no longer needs to use weights from the unpruned network (see Section 3.3). In addition, some NAS techniques can also be applied to the pruning approach including borrowing trained coefficients and reinforcement learning search.

Typically, compression is evaluated on large data-sets such as the ILSVRC-2012 dataset with one thousand object categories. In practice, resource constraints in embedded devices don't allow a large capacity of optimized networks. Compressing a model to best fit a constrained environment should consider but not be limited to the deployment environment, target device, speed/compression trade-offs, and accuracy requirements [29].

Based on the reviewed pruning techniques, we recommend the following for effective pruning:

- Uniform pruning introduces accuracy loss therefore setting the pruning ratio to vary by layers is better [159].
- Dynamic pruning may result in higher accuracy and maintain higher network capacity [246].
- Structurally pruning a network may benefit from maturing libraries especially when pruning at a high level [241].
- Training a pruned model from scratch sometimes, but not always (see Section 3.3), is more efficient than tunning from the unpruned weights [160].
- Penalty-based pruning typically reduces accuracy loss compared with magnitude-based pruning [255]. However, recent efforts are narrowing the gap [72].

5.2. Quantization

Section 4 discusses quantization techniques. It describes binarized quantized neural networks, and reduced precision networks, along with their training methods. We described low-bit dataset validation techniques and results. We also list the accuracy of popular quantization frameworks and described hardware implementations in Section 4.3.

Quantization usually results in a loss of accuracy due to information lost during the quantization process. This is particularly evident on compact networks. Most of the early low bit quantization approaches only compare performance on small datasets (e.g., MNIST, and CIFAR-10) [58,94,156,200,235,269]. However, observations showed that some quantized networks could outperform the original network (see: Section 4.4). Additionally, non-uniform distribution data may lead to further deterioration in quantization performance [275]. Sometimes this can be ameliorated by normalization in fine-tuning [172] or by non-linear quantization (e.g., log representation) [175].

Advanced quantization techniques have improved accuracy. Asymmetric quantization [120] maintains higher dynamic range by using a zero point in addition to a regular scale parameter. Overheads introduced by the zero point were minimized by pipelining the processing unit. Calibration based quantization [173] removed zero points and replaced them with precise scales obtained from a calibrating dataset. Quantization-aware training was shown to further improve quantization accuracy.

8-bit quantization is widely applied in practice as a good trade-off between accuracy and compression. It can easily be deployed on current processors and custom hardware. Minimal accuracy loss is experienced especially when quantization-aware training is enabled. Binarized networks have also achieved reasonable accuracy with specialized hardware designs.

Though BN has advantages to help training and pruning, an issue with BN is that it may require a large dynamic range across a single layer kernel or between different channels. This may make layer-wise quantization more difficult. Because of this per channel quantization is recommended [131].

To achieve better accuracy following quantization, we recommend:

- Use asymmetrical quantization. It preserves flexibility over the quantization range even though it has computational overheads [120].
- Quantize the weights rather than the activations. Activation is more sensitive to numerical precision [75].
- Do not quantize biases. They do not require significant storage. High precision biases in all layers [114], and first/last layers [200,272], maintain higher network accuracy.
- Quantize kernels channel-wise instead of layer-wise to significantly improve accuracy [131].
- Fine-tune the quantized model. It reduces the accuracy gap between the quantized model and the real-valued model [244].
- Initially train using a 32-bit floating point model. Low-bit quantized model can be difficult to train from scratch - especially compact models on large-scaled data-sets [272].
- The sensitivity of quantization is ordered as gradients, activations, and then weights [272].
- Stochastic quantization of gradients is necessary when training quantized models [89,272].

6. Future Work

Although punning and quantization algorithms help reduce the computation cost and bandwidth burden, there are still areas for improvement. In this section we highlight future work to further improvement quantization and prunning.

Automatic Compression. Low bit width quantization can cause significant accuracy loss, especially when the quantized bit-width is very narrow and the dataset is large [272,155]. Automatic quantization is a technique to automatically search quantization encoding to evaluate accuracy loss verses compression ratio. Similarly, automatic prunning is a technique to automatically search different prunning approaches to evaluate the sparsity ratio versus accuracy. Similar to hyperparameter tuning [257], this can be performed without human intervention using any number of search techniques (e.g., random search, genetic search, etc.).

Compression on Other Types of Neural Networks. Current compression research is primarily focused on CNNs. More specifically, research is primarily directed towards CNN classification tasks. Future work should also consider other types of applications such as object detection, speech recognition, language translation, etc. Network compression verses accuracy for different applications is an interesting area of research.

Hardware Adaptation. Hardware implementations may limit the effectiveness of pruning algorithms. For example, element-wise pruning barely reduce computations or bandwidth when using im2col-gemm on general-purpose processors [264]. Similarly, shape-wise pruning is not typically able to be implemented on dedicated CNN accelerators. Hardware-software co-design of compression techniques for hardware accelerators should be considered to achieve the best system efficiency.

Global Methods. Network optimizations are typically applied separately without information from one optimization informing any other optimization. Recently, approaches that consider optimization effectiveness at multiple layers have been proposed. [150] discusses pruning combined with tensor factorization that results in better overall compression. Similar techniques can be considered using different types and levels of compression and factorization.

7. Conclusions

Deep neural networks have been applied in many applications exhibiting extraordinary abilities in the field of computer vision. However, complex network architectures challenge efficient real-time deployment and require significant computation resources and energy costs. These challenges can be overcome through optimizations such as network compression. Network compression can often be realized with little loss of accuracy. In some cases accuracy may even improve.

Pruning can be categorized as static (3.1) if it is performed offline or dynamic (3.2) if it is performed at run-time. The criteria applied to removing redundant computations if often just a simple magnitude of weights with values near zero being pruned. More

complicated methods include checking the l_p-norm. Techniques such as LASSO and Ridge are built around l_1 and l_2 norms. Pruning can be performed element-wise, channel-wise, shape-wise, filter-wise, layer-wise and even network-wise. Each has trade-offs in compression, accuracy, and speedup.

Table 4
Quantization Network Performance on ILSVRC2012 for various bit-widths of the weights W and activation A (aka. feature)

Model	Deployment	Bit-width		Acc. Drop		Ref.
		W	A	Top-1	Top-5	
AlexNet	QuantNet	1	32	−1.70%	−1.50%	[253]
	BWNH	1	32	−1.40%	−0.70%	[107]
	SYQ	2	8	−1.00%	−0.60%	[66]
	TSQ	2	2	−0.90%	−0.30%	[239]
	INQ	5	32	−0.87%	−1.39%	[269]
	PACT	4	3	−0.60%	−1.00%	[44]
	QIL	4	4	−0.20%	-	[127]
	Mixed-Precision	16	16	−0.16%	-	[172]
	PACT	32	5	−0.10%	−0.20%	[44]
	QIL	5	5	−0.10%	-	[127]
	QuantNet	3(±4)	32	−0.10%	−0.10%	[253]
	ELNN	3(±4)	32	0.00%	0.20%	[144]
	DoReFa-Net	32	3	0.00%	−0.90%	[272]
	TensorRT	8	8	0.03%	0.00%	[173]
	PACT	2	2	0.10%	−0.70%	[44]
	PACT	32	2	0.20%	−0.20%	[44]
	DoReFa-Net	32	5	0.20%	−0.50%	[272]
	QuantNet	3(±2)	32	0.30%	0.00%	[253]
	DoReFa-Net	32	4	0.30%	−0.50%	[272]
	WRPN	2	32	0.40%	-	[174]
	DFP16	16	16	0.49%	0.59%	[54]
	PACT	3	2	0.50%	−0.10%	[44]
	PACT	4	2	0.50%	−0.10%	[44]
	SYQ	1	8	0.50%	0.80%	[66]
	QIL	3	3	0.50%	-	[127]
	FP8	8	8	0.50%	-	[237]
	BalancedQ	32	2	0.60%	−2.00%	[273]
	ELNN	3(±2)	32	0.80%	0.60%	[144]
	SYQ	1	4	0.90%	0.80%	[66]
	QuantNet	2	32	0.90%	0.30%	[253]
	FFN	2	32	1.00%	0.30%	[238]
	DoReFa-Net	32	2	1.00%	0.10%	[272]
	Unified INT8	8	8	1.00%	-	[275]
	DeepShift-PS	6	32	1.19%	0.67%	[62]
	WEQ	4	4	1.20%	1.00%	[192]
	LQ-NETs	2	32	1.30%	0.80%	[262]
	SYQ	2	2	1.30%	1.00%	[66]
	LQ-NETs	1	2	1.40%	1.40%	[262]
	BalancedQ	2	2	1.40%	−1.00%	[273]
	WRPN-2x	8	8	1.50%	-	[174]
	DoReFa-Net	1	4	1.50%	-	[272]
	DeepShift-Q	6	32	1.55%	0.81%	[62]
	WRPN-2x	32	8	1.60%	-	[174]
	WEQ	3	4	1.60%	1.10%	[192]
	WRPN-2x	8	4	1.70%	-	[174]
	WRPN-2x	4	8	1.70%	-	[174]
	SYQ	1	2	1.70%	1.60%	[66]
	ELNN	2	32	1.80%	1.80%	[144]
	WRPN-2x	4	4	1.90%	-	[174]
	WRPN-2x	32	4	1.90%	-	[174]
GoogLeNet	Mixed-Precision	16	16	−0.10%	-	[172]
	DeepShift-PS	6	32	−0.09%	−0.09%	[62]
	DFP16	16	16	−0.08%	0.00%	[54]
	AngleEye	16	16	0.05%	0.45%	[85]
	AngleEye	16	16	0.05%	0.45%	[85]
	ShiftCNN	3	4	0.05%	0.09%	[84]
	DeepShift-Q	6	32	0.27%	0.29%	[62]
	LogQuant	32	6	0.36%	0.28%	[30]
	ShiftCNN	2	4	0.39%	0.29%	[84]
	TensorRT	8	8	0.45%	0.19%	[173]
	LogQuant	6	32	0.64%	0.67%	[30]
	INQ	5	32	0.76%	0.25%	[269]
	ELNN	3(±4)	32	2.40%	1.40%	[144]
	ELNN	3(±2)	32	2.80%	1.60%	[144]
	LogQuant	6	6	3.43%	0.78%	[30]
	QNN	4	4	5.10%	7.80%	[113]
	QNN	6	6	5.20%	8.10%	[113]

(*continued on next page*)

Table 4 (*continued*)

Model	Deployment	Bit-width		Acc. Drop		Ref.
		W	A	Top-1	Top-5	
	ELNN	2	32	5.60%	3.50%	[144]
	BWN	1	32	5.80%	4.80%	[200]
	AngleEye	8	8	6.00%	3.20%	[85]
	TWN	2	32	7.50%	4.80%	[146]
	ELNN	1	32	8.40%	5.70%	[144]
	BWN	2	32	9.70%	6.50%	[200]
	ShiftCNN	1	4	11.26%	7.36%	[84]
	LogQuant	32	3	13.50%	8.93%	[30]
	LogQuant	3	3	18.07%	12.85%	[30]
	LogQuant	4	4	18.57%	13.21%	[30]
	LogQuant	32	4	18.57%	13.21%	[30]
	BNN	1	1	24.20%	20.90%	[53]
	AngleEye	6	6	52.10%	57.35%	[85]
MobileNet V1	HAQ-Cloud	6	6	−0.38%	−0.23%	[236]
	HAQ-Edge	6	6	−0.38%	−0.34%	[236]
	MelinusNet59	1	1	−0.10%	-	[21]
	HAQ-Edge	5	5	0.24%	0.08%	[236]
	PACT	6	6	0.36%	0.26%	[44]
	PACT	6	6	0.36%	0.26%	[44]
	HAQ-Cloud	5	5	0.85%	0.48%	[236]
	HAQ-Edge	4	4	3.42%	1.95%	[236]
	PACT	5	5	3.82%	2.20%	[44]
	PACT	5	5	3.82%	2.20%	[44]
	HAQ-Cloud	4	4	5.49%	3.25%	[236]
	PACT	4	4	8.38%	5.66%	[44]
MobileNet V2	HAQ-Edge	6	6	−0.08%	−0.11%	[236]
	HAQ-Cloud	6	6	−0.04%	0.01%	[236]
	Unified INT8	8	8	0.00%	-	[275]
	PACT	6	6	0.56%	0.25%	[44]
	HAQ-Edge	5	5	0.91%	0.34%	[236]
	HAQ-Cloud	5	5	2.36%	1.31%	[236]
	PACT	5	5	2.97%	1.67%	[44]
	HAQ-Cloud	4	4	4.80%	2.79%	[236]
	HAQ-Edge	4	4	4.82%	2.92%	[236]
	PACT	4	4	10.42%	6.53%	[44]
ResNet-18	RangeBN	8	8	−0.60%	-	[15]
	LBM	8	8	−0.60%	-	[268]
	QuantNet	5	32	−0.30%	−0.10%	[253]
	QIL	5	5	−0.20%	-	[127]
	QuantNet	3(±4)	32	−0.10%	−0.10%	[253]
	ShiftCNN	3	4	0.03%	0.12%	[84]
	LQ-NETs	4	32	0.20%	0.50%	[262]
	QIL	3	32	0.30%	0.30%	[127]
	LPBN	32	5	0.30%	0.40%	[31]
	QuantNet	3(±2)	32	0.40%	0.20%	[253]
	PACT	32	4	0.40%	0.30%	[44]
	SeerNet	4	1	0.42%	0.18%	[32]
	ShiftCNN	2	4	0.54%	0.34%	[84]
	PACT	5	5	0.60%	0.30%	[44]
	INQ	4	32	0.62%	0.10%	[269]
	Unified INT8	8	8	0.63%	-	[275]
	QIL	5	5	0.80%	-	[127]
	LQ-NETs	3(±4)	32	0.90%	0.80%	[262]
	QIL	3	3	1.00%	-	[127]
	DeepShift-Q	6	32	1.09%	0.47%	[62]
	ELNN	3(±2)	32	1.10%	0.70%	[144]
	PACT	32	3	1.20%	0.70%	[44]
	PACT	4	4	1.20%	0.60%	[44]
	QuantNet	2	32	1.20%	0.60%	[253]
	ELNN	3(±4)	32	1.30%	0.60%	[144]
	DeepShift-PS	6	32	1.44%	0.67%	[62]
	ABC-Net	5	32	1.46%	1.18%	[155]
	ELNN	3(±2)	32	1.60%	1.10%	[144]
	DoReFa-Net	32	5	1.70%	1.00%	[272]
	SYQ	2	8	1.90%	1.40%	[66]
	DoReFa-Net	32	4	1.90%	1.10%	[272]
	LQ-NETs	3	3	2.00%	1.60%	[262]
	DoReFa-Net	5	5	2.00%	1.30%	[272]
	ELNN	2	32	2.10%	1.50%	[144]
	QIL	2	32	2.10%	1.30%	[127]
	DoReFa-Net	32	3	2.10%	1.40%	[272]
	QIL	4	4	2.20%	-	[127]

Table 4 (*continued*)

Model	Deployment	Bit-width		Acc. Drop		Ref.
		W	A	Top-1	Top-5	
	LQ-NETs	2	32	2.20%	1.60%	[262]
	GroupNet-8	1	1	2.20%	1.40%	[276]
	PACT	3	3	2.30%	1.40%	[44]
	DoReFa-Net	4	4	2.30%	1.50%	[272]
	TTN	2	32	2.50%	1.80%	[274]
	TTQ	2	32	2.70%	2.00%	[277]
	AddNN	32	32	2.80%	1.50%	[35]
	ELNN	2	32	2.80%	1.50%	[144]
	LPBN	32	4	2.90%	1.70%	[31]
	PACT	32	2	2.90%	2.00%	[44]
	DoReFa-Net	3	3	2.90%	2.00%	[272]
	QuantNet	1	32	3.10%	1.90%	[253]
	INQ	2	32	3.10%	1.90%	[269]
ResNet-34	WRPN-2x	4	4	−0.93%	-	[174]
	WRPN-2x	4	8	−0.89%	-	[174]
	QIL	4	4	0.00%	-	[127]
	QIL	5	5	0.00%	-	[127]
	WRPN-2x	4	2	0.01%	-	[174]
	WRPN-2x	2	4	0.09%	-	[174]
	WRPN-2x	2	2	0.27%	-	[174]
	SeerNet	4	1	0.35%	0.17%	[32]
	Unified INT8	8	8	0.39%	-	[275]
	LCCL			0.43%	0.17%	[59]
	QIL	3	3	0.60%	-	[127]
	WRPN-3x	1	1	0.90%	-	[174]
	WRPN-3x	1	1	1.21%	-	[174]
	GroupNet-8	1	1	1.40%	1.00%	[276]
	dLAC	2	16	1.67%	0.89%	[235]
	LQ-NETs	3	3	1.90%	1.20%	[262]
	GroupNet**-5	1	1	2.70%	2.10%	[276]
	IR-Net	1	32	2.90%	1.80%	[199]
	QIL	2	2	3.10%	-	[127]
	WRPN-2x	1	1	3.40%	-	[174]
	WRPN-2x	1	1	3.74%	-	[174]
	LQ-NETs	2	2	4.00%	2.30%	[262]
	GroupNet-5	1	1	4.70%	3.40%	[276]
	ABC-Net	5	5	4.90%	3.10%	[155]
	HWGQ	1	32	5.10%	3.40%	[31]
	WAGEUBN	8	8	5.18%	-	[254]
	ABC-Net	3	3	6.60%	3.90%	[155]
	LQ-NETs	1	2	6.70%	4.40%	[262]
	LQ-NETs	4	4	6.70%	4.40%	[262]
	BCGD	1	4	7.60%	4.70%	[256]
	HWGQ	1	2	9.00%	5.60%	[31]
	IR-Net	1	1	9.50%	6.20%	[199]
	CI-BCNN (add)	1	1	11.07%	6.39%	[240]
	Bi-Real	1	1	11.10%	7.40%	[252]
	WRPN-1x	1	1	12.80%	-	[174]
	WRPN	1	1	13.05%	-	[174]
	CI-BCNN	1	1	13.59%	8.65%	[240]
	DoReFa-Net	1	4	14.60%	-	[272]
	DoReFa-Net	1	2	20.40%	-	[272]
	ABC-Net	1	1	20.90%	14.80%	[155]
	BNN	1	1	29.10%	24.20%	[272]
ResNet-50	Mixed-Precision	16	16	−0.12%	-	[172]
	DFP16	16	16	−0.07%	−0.06%	[54]
	QuantNet	5	32	0.00%	0.00%	[253]
	LQ-NETs	4	32	0.00%	0.10%	[262]
	FGQ	32	32	0.00%	-	[170]
	TensorRT	8	8	0.13%	0.12%	[173]
	PACT	5	5	0.20%	−0.20%	[44]
	QuantNet	3(±4)	32	0.20%	0.00%	[253]
	Unified INT8	8	8	0.26%	-	[275]
	ShiftCNN	3	4	0.29%	0.15%	[84]
	ShiftCNN	3	4	0.31%	0.16%	[84]
	PACT	4	4	0.40%	−0.10%	[44]
	LPBN	32	5	0.40%	0.40%	[81]
	ShiftCNN	2	4	0.67%	0.41%	[84]
	DeepShift-Q	6	32	0.81%	0.21%	[62]
	DeepShift-PS	6	32	0.84%	0.31%	[62]

(*continued on next page*)

Table 4 (*continued*)

Model	Deployment	Bit-width		Acc. Drop		Ref.
		W	A	Top-1	Top-5	
	PACT	5	32	0.90%	0.20%	[44]
	QuantNet	3(±2)	32	0.90%	0.40%	[253]
	PACT	4	32	1.00%	0.20%	[44]
	dLAC	2	16	1.20%	-	[235]
	QuantNet	2	32	1.20%	0.60%	[253]
	AddNN	32	32	1.30%	1.20%	[35]
	LQ-NETs	4	4	1.30%	0.80%	[262]
	LQ-NETs	2	32	1.30%	0.90%	[262]
	INQ	5	32	1.32%	0.41%	[269]
	PACT	3	32	1.40%	0.50%	[44]
	IAO	8	8	1.50%	-	[120]
	PACT	3	3	1.60%	0.50%	[44]
	HAQ	2MP	4MP	1.91%	-	[236]
	HAQ	MP	MP	2.09%	-	[236]
	LQ-NETs	3	3	2.20%	1.60%	[262]
	LPBN	32	4	2.20%	1.20%	[81]
	Deep Comp.	3	MP	2.29%	-	[92]
	PACT	4	2	2.40%	1.20%	[44]
	ShiftCNN	2	4	2.49%	1.64%	[84]
	FFN	2	32	2.50%	1.30%	[238]
	UNIQ	4	8	2.60%	-	[18]
	QuantNet	1	32	3.20%	1.70%	[253]
	SYQ	2	8	3.70%	2.10%	[66]
	FGQ-TWN	2	8	4.29%	-	[170]
	PACT	2	2	4.70%	2.60%	[44]
	LQ-NETs	2	2	4.90%	2.90%	[262]
	SYQ	1	8	5.40%	3.40%	[66]
	DoReFa-Net	4	4	5.50%	3.30%	[272]
	DoReFa-Net	5	5	5.50%	−0.20%	[272]
	FGQ	2	8	5.60%	-	[170]
	ABC-Net	5	5	6.30%	3.50%	[155]
	FGQ-TWN	2	4	6.67%	-	[170]
	HWGQ	1	2	6.90%	4.60%	[31]
ResNet-100	IAO	8	8	1.40%	-	[120]
ResNet-101	TensorRT	8	8	−0.01%	0.05%	[173]
	FGQ-TWN	2	8	3.65%	-	[170]
	FGQ-TWN	2	4	6.81%	-	[170]
ResNet-150	IAO	8	8	2.10%	-	[120]
ResNet-152	TensorRT	8	8	0.08%	0.04%	[173]
	dLAC	2	16	1.20%	0.64%	[235]
SqueezeNet	AngleEye	16	16	0.00%	0.01%	[85]
	ShiftCNN	3	4	0.01%	0.01%	[84]
	ShiftCNN	2	4	1.01%	0.71%	[84]
	AngleEye	8	8	1.42%	1.05%	[85]
	AngleEye	6	6	28.13%	27.43%	[85]
	ShiftCNN	1	4	35.39%	35.09%	[84]
VGG-16	ELNN	3(±4)	32	−1.10%	−1.00%	[144]
	ELNN	3(±2)	32	−0.60%	−0.80%	[144]
	AngleEye	16	16	0.09%	−0.05%	[85]
	DFP16	16	16	0.11%	0.29%	[54]
	AngleEye	8	8	0.21%	0.08%	[85]
	SeerNet	4	1	0.28%	0.10%	[32]
	DeepShift-Q	6	32	0.29%	0.11%	[62]
	FFN	2	32	0.30%	−0.20%	[238]
	DeepShift-PS	6	32	0.47%	0.30%	[62]
	DeepShift-Q	6	32	0.72%	0.29%	[62]
	INQ	5	32	0.77%	0.08%	[62]
	TWN	2	32	1.10%	0.30%	[146]
	ELNN	2	32	2.00%	0.90%	[144]
	TSQ	2	2	2.00%	0.70%	[239]
	AngleEye	16	16	2.15%	1.49%	[85]
	BWN	2	32	2.20%	1.20%	[200]
	AngleEye	8	8	2.35%	1.76%	[85]
	ELNN	1	32	3.30%	1.80%	[144]
	AngleEye	6	6	9.07%	6.58%	[85]
	AngleEye	6	6	22.38%	17.75%	[85]
	LogQuant	3	3	-	0.99%	[30]
	LogQuant	4	4	-	0.51%	[30]
	LogQuant	6	6	-	0.83%	[30]
	LogQuant	32	3	-	0.82%	[30]

Table 4 (*continued*)

Model	Deployment	Bit-width		Acc. Drop		Ref.
		W	A	Top-1	Top-5	
	LogQuant	32	4	-	0.36%	[30]
	LogQuant	32	6	-	0.31%	[30]
	LogQuant	6	32	-	0.76%	[30]
	LDR	5	4	-	0.90%	[175]
	LogNN	5	4	-	1.38%	[175]

Quantization reduces computations by reducing the precision of the datatype. Most networks are trained using 32-bit floating point. Weights, biases, and activations may then be quantized typically to 8-bit integers. Lower bit width quantizations have been performed with single bit being termed a binary neural network. It is difficult to (re) train very low bit width neural networks. A single bit is not differentiable thereby prohibiting back propagation. Lower bit widths cause difficulties for computing gradients. The advantage of quantization is significantly improved performance (usually 2-3x) and dramatically reduced storage requirements. In addition to describing how quantization is performed we also included an overview of popular libraries and frameworks that support quantization. We further provided a comparison of accuracy for a number of networks using different frameworks Table 2.

In this paper, we summarized pruning and quantization techniques. Pruning removes redundant computations that have limited contribution to a result. Quantization reduces computations by reducing the precision of the datatype. Both can be used independently or in combination to reduce storage requirements and accelerate inference.

8. Quantization Performance Results

CRediT authorship contribution statement

Tailin Liang: Writing - original draft, Visualization, Formal analysis, Resources, Funding acquisition. **John Glossner:** Conceptualization, Writing - review & editing, Supervision, Resources. **Lei Wang:** Resources, Investigation, Validation, Project administration, Funding acquisition. **Shaobo Shi:** Resources, Investigation, Validation, Funding acquisition. **Xiaotong Zhang:** Conceptualization, Supervision, Resources, Visualization.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, in: A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng (Eds.), arXiv preprint arXiv: 1603.04467, 2016.

[2] Abdel-Hamid, O., Mohamed, A.r., Jiang, H., Deng, L., Penn, G., Yu, D., 2014. Convolutional Neural Networks for Speech Recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing 22, 1533–1545. http://ieeexplore.ieee.org/document/6857341/, 10.1109/TASLP.2014.2339736. .

[3] Abdelouahab, K., Pelcat, M., Serot, J., Berry, F., 2018. Accelerating CNN inference on FPGAs: A Survey. ArXiv preprint http://arxiv.org/abs/1806.01683. .

[4] Achronix Semiconductor Corporation, 2020. FPGAs Enable the Next Generation of Communication and Networking Solutions. White Paper WP021, 1–15. .

[5] Albanie, 2020. convnet-burden. https://github.com/albanie/convnet-burden. .

[6] Alemdar, H., Leroy, V., Prost-Boucle, A., Petrot, F., 2017. Ternary neural networks for resource-efficient AI applications, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 2547–2554. https://ieeexplore.ieee.org/abstract/document/7966166/, DOI: 10.1109/IJCNN.2017.7966166. .

[7] AMD, Radeon Instinct MI25 Accelerator. https://www.amd.com/en/products/professional-graphics/instinct-mi25. .

[8] Arm, 2015. ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile. https://developer.arm.com/documentation/ddi0487/latest. https://developer.arm.com/documentation/ddi0487/latest. .

[9] Arm, 2020. Arm Cortex-M Processor Comparison Table. https://developer.arm.com/ip-products/processors/cortex-a. .

[10] Arm, Graphics, C., 2020. MALI-G76 High-Performance GPU for Complex Graphics Features and Bene ts High Performance for Mixed Realities. https://www.arm.com/products/silicon-ip-multimedia/gpu/mali-g76. .

[11] ARM, Reddy, V.G., 2008. Neon technology introduction. ARM Corporation, 1–34http://caxapa.ru/thumbs/301908/AT_-_NEON_for_Multimedia_Applications.pdf. .

[12] M.G. Augasta, T. Kathirvalavakumar, Pruning algorithms of neural networks - A comparative study, Open Computer Science 3 (2013) 105–115, https://doi.org/10.2478/s13537-013-0109-x.

[13] Baidu, 2019. PArallel Distributed Deep LEarning: Machine Learning Framework from Industrial Practice. https://github.com/PaddlePaddle/Paddle. .

[14] W. Balzer, M. Takahashi, J. Ohta, K. Kyuma, Weight quantization in Boltzmann machines, Neural Networks 4 (1991) 405–409, https://doi.org/10.1016/0893-6080(91)90077-I.

[15] R. Banner, I. Hubara, E. Hoffer, D. Soudry, Scalable methods for 8-bit training of neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2018, pp. 5145–5153, http://papers.nips.cc/paper/7761-scalable-methods-for-8-bit-training-of-neural-networks.

[16] R. Banner, Y. Nahshan, D. Soudry, Post training 4-bit quantization of convolutional networks for rapid-deployment, in: Advances in Neural Information Processing Systems (NIPS), 2019, pp. 7950–7958.

[17] Baoyuan Liu, Min Wang, H. Foroosh, M. Tappen, M. Penksy, Sparse Convolutional Neural Networks, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 806–814, https://doi.org/10.1109/CVPR.2015.7298681.

[18] Baskin, C., Schwartz, E., Zheltonozhskii, E., Liss, N., Giryes, R., Bronstein, A.M., Mendelson, A., 2018. UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks. arXiv preprint arXiv:1804.10969 http://arxiv.org/abs/1804.10969. .

[19] Bengio, E., Bacon, P.L., Pineau, J., Precup, D., 2015. Conditional Computation in Neural Networks for faster models. ArXiv preprint http://arxiv.org/abs/1511.06297. .

[20] Bengio, Y., 2013. Estimating or Propagating Gradients Through Stochastic Neurons. ArXiv preprint http://arxiv.org/abs/1305.2982. .

[21] Bethge, J., Bartz, C., Yang, H., Chen, Y., Meinel, C., 2020. MeliusNet: Can Binary Neural Networks Achieve MobileNet-level Accuracy? ArXiv preprint http://arxiv.org/abs/2001.05936. .

[22] J. Bethge, H. Yang, M. Bornstein, C. Meinel, BinaryDenseNet: Developing an architecture for binary neural networks, in: Proceedings - 2019 International Conference on Computer Vision Workshop ICCVW 2019, 1951–196010.1109/ICCVW.2019.00244, 2019.

[23] S. Bianco, R. Cadene, L. Celona, P. Napoletano, Benchmark analysis of representative deep neural network architectures, IEEE Access 6 (2018) 64270–64277, https://doi.org/10.1109/ACCESS.2018.2877890.

[24] Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J., 2020. What is the State of Neural Network Pruning? ArXiv preprint http://arxiv.org/abs/2003.03033. .

[25] T. Bolukbasi, J. Wang, O. Dekel, V. Saligrama, Adaptive Neural Networks for Efficient Inference, in: Thirty-fourth International Conference on Machine Learning, 2017.

[26] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020. Language Models are Few-Shot Learners. ArXiv preprint http://arxiv.org/abs/2005.14165. .

[27] Bucilua, C., Caruana, R., Niculescu-Mizil, A., 2006. Model compression, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge

discovery and data mining - KDD '06, ACM Press, New York, New York, USA. p. 535. https://dl.acm.org/doi/abs/10.1145/1150402.1150464, DOI: 10.1145/1150402.1150464. .
[28] BUG1989, 2019. BUG1989/caffe-int8-convert-tools: Generate a quantization parameter file for ncnn framework int8 inference. https://github.com/BUG1989/caffe-INT8-convert-tools. .
[29] Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S., 2019. Once-for-All: Train One Network and Specialize it for Efficient Deployment. ArXiv preprint, 1–15 http://arxiv.org/abs/1908.09791. .
[30] J. Cai, M. Takemoto, H. Nakajo, A Deep Look into Logarithmic Quantization of Model Parameters in Neural Networks, in: Proceedings of the 10th International Conference on Advances in Information Technology - IAIT 2018, ACM Press, New York, New York, USA, 2018, pp. 1–8, https://doi.org/10.1145/3291280.3291800.
[31] Z. Cai, X. He, J. Sun, N. Vasconcelos, Deep Learning with Low Precision by Half-Wave Gaussian Quantization, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 5406–5414, https://doi.org/10.1109/CVPR.2017.574.
[32] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, Z. Yang, SeerNet: Predicting Convolutional Neural Network Feature-Map Sparsity through Low-Bit Quantization, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
[33] Carreira-Perpinan, M.A., Idelbayev, Y., 2018. Learning-Compression Algorithms for Neural Net Pruning, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 8532–8541. https://ieeexplore.ieee.org/document/8578988/, DOI: 10.1109/CVPR.2018.00890. .
[34] Chellapilla, K., Puri, S., Simard, P., 2006. High Performance Convolutional Neural Networks for Document Processing, in: Tenth International Workshop on Frontiers in Handwriting Recognition. https://hal.inria.fr/inria-00112631/, 10.1.1.137.482. .
[35] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, C. Xu, AdderNet: Do We Really Need Multiplications in Deep Learning?, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp 1468–1477.
[36] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A., 2018. TVM: An automated end-to-end optimizing compiler for deep learning, in: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, pp. 579–594. http://arxiv.org/abs/1802.04799. .
[37] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: International Conference on Machine Learning, 2015, pp. 2285–2294.
[38] Y. Chen, T. Chen, Z. Xu, N. Sun, O. Temam, DianNao family: Energy-Efficient Hardware Accelerators for Machine Learning, Commun. ACM 59 (2016) 105–112, https://doi.org/10.1145/2594446 http://dl.acm.org/citation.cfm?doid=3013530.2996864, 10.1145/2996864.
[39] Cheng, J., Wang, P.s., Li, G., Hu, Q.h., Lu, H.q., 2018. Recent advances in efficient computation of deep convolutional neural networks. Frontiers of Information Technology & Electronic Engineering 19, 64–77. http://link.springer.com/10.1631/FITEE.1700789, 10.1631/FITEE.1700789 .
[40] Cheng, Y., Wang, D., Zhou, P., Zhang, T., 2017. A Survey of Model Compression and Acceleration for Deep Neural Networks. ArXiv preprint http://arxiv.org/abs/1710.09282. .
[41] Cheng, Z., Soudry, D., Mao, Z., Lan, Z., 2015. Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation. ArXiv preprint http://cn.arxiv.org/pdf/1503.03562.pdf http://arxiv.org/abs/1503.03562. .
[42] Z. Chiliang, H. Tao, G. Yingda, Y. Zuochang, Accelerating Convolutional Neural Networks with Dynamic Channel Pruning, in: 2019 Data Compression Conference (DCC), IEEE, 2019, p. 563, https://doi.org/10.1109/DCC.2019.00075.
[43] B. Choi, J.H. Lee, D.H. Kim, Solving local minima problem with large number of hidden nodes on two-layered feed-forward artificial neural networks, Neurocomputing 71 (2008) 3640–3643, https://doi.org/10.1016/j.neucom.2008.04.004.
[44] Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.j., Srinivasan, V., Gopalakrishnan, K., 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. ArXiv preprint, 1–15 http://arxiv.org/abs/1805.06085. .
[45] Y. Choi, M. El-Khamy, J. Lee, Towards the Limit of Network Quantization, in: International Conference on Learning Representations(ICLR), IEEE, 2017.
[46] Y. Choi, S.S. Member, D. Bae, J. Sim, S.S. Member, S. Choi, M. Kim, S.S. Member, L.s.S. Kim, S.S. Member, Energy-Efficient Design of Processing Element for Convolutional Neural Network, IEEE Trans. Circuits Syst. II Express Briefs 64 (2017) 1332–1336, https://doi.org/10.1109/TCSII.2017.2691771, http://ieeexplore.ieee.org/document/7893765/.
[47] Chollet, F., Google, C., 2017. Xception: Deep Learning with Depthwise Separable Convolutions, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 1251–1258. http://ieeexplore.ieee.org/document/8099678/, DOI: 10.1109/CVPR.2017.195. .
[48] T. Choudhary, V. Mishra, A. Goswami, J. Sarangapani, A comprehensive survey on model compression and acceleration, Artif. Intell. Rev. 53 (2020) 5113–5155, https://doi.org/10.1007/s10462-020-09816-7.
[49] Cornea, M., 2015. Intel AVX-512 Instructions and Their Use in the Implementation of Math Functions. Intel Corporation. .
[50] Cotofana, S., Vassiliadis, S., Logic, T., Addition, B., Addition, S., 1997. Low Weight and Fan-In Neural Networks for Basic Arithmetic Operations, in: 15th IMACS World Congress, pp. 227–232. 10.1.1.50.4450. .
[51] Courbariaux, M., Bengio, Y., David, J.P., 2014. Training deep neural networks with low precision multiplications, in: International Conference on Learning Representations(ICLR), pp. 1–10. http://arxiv.org/abs/1412.7024, arXiv: 1412.7024. .
[52] Courbariaux, M., Bengio, Y., David, J.P., 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations, in: Advances in Neural Information Processing Systems (NIPS), pp. 1–9. http://arxiv.org/abs/1511.00363, DOI: 10.5555/2969442.2969588. .
[53] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. ArXiv preprint https://github.com/MatthieuCourbariaux/ http://arxiv.org/abs/1602.02830. .
[54] Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., Heinecke, A., Dubey, P., Corbal, J., Shustrov, N., Dubtsov, R., Fomenko, E., Pirogov, V., 2018. Mixed Precision Training of Convolutional Neural Networks using Integer Operations, in: International Conference on Learning Representations(ICLR), pp. 1–11. https://www.anandtech.com/show/11741/hot-chips-intel-knights-mill-live-blog-445pm-pt-1145pm-utc http://arxiv.org/abs/1802.00930. .
[55] M. Dash, H. Liu, Feature selection for classification, Intelligent Data Analysis 1 (1997) 131–156, https://doi.org/10.3233/IDA-1997-1302.
[56] A. Davis, I. Arel, Low-Rank Approximations for Conditional Feedforward Computation in Deep Neural Networks, International Conference on Learning Representations Workshops (ICLRW) (2013) 1–10, http://arxiv.org/abs/1312.4461.
[57] Deng, W., Yin, W., Zhang, Y., 2013. Group sparse optimization by alternating direction method, in: Van De Ville, D., Goyal, V.K., Papadakis, M. (Eds.), Wavelets and Sparsity XV, p. 88580R. http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2024410, DOI: 10.1117/12.2024410. .
[58] T. Dettmers, 8-Bit Approximations for Parallelism in Deep Learning, in: International Conference on Learning Representations(ICLR), 2015.
[59] X. Dong, J. Huang, Y. Yang, S. Yan, More is less: A more complicated network with less inference complexity, in: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition CVPR 2017 2017-Janua, 1895–1903, 2017, https://doi.org/10.1109/CVPR.2017.205.
[60] J.J. Dongarra, J. Du Croz, S. Hammarling, I.S. Duff, A set of level 3 basic linear algebra subprograms, ACM Transactions on Mathematical Software (TOMS) 16 (1990) 1–17, https://doi.org/10.1145/77626.79170.
[61] Dukhan, M., Yiming, W., Hao, L., Lu, H., 2019. QNNPACK: Open source library for optimized mobile deep learning - Facebook Engineering. https://engineering.fb.com/ml-applications/qnnpack/. .
[62] Elhoushi, M., Chen, Z., Shafiq, F., Tian, Y.H., Li, J.Y., 2019. DeepShift: Towards Multiplication-Less Neural Networks. ArXiv preprint http://arxiv.org/abs/1905.13298. .
[63] T. Elsken, J.H. Metzen, F. Hutter, Neural Architecture Search, J. Mach. Learn. Res. 20 (2019) 63–77, http://link.springer.com/10.1007/978-3-030-05318-5_3.
[64] A.P. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, IEEE Trans. Neural Networks 12 (2001) 1386–1389, https://doi.org/10.1109/72.963775.
[65] Esser, S.K., Merolla, P.A., Arthur, J.V., Cassidy, A.S., Appuswamy, R., Andreopoulos, A., Berg, D.J., McKinstry, J.L., Melano, T., Barch, D.R., di Nolfo, C., Datta, P., Amir, A., Taba, B., Flickner, M.D., Modha, D.S., 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. Proceedings of the National Academy of Sciences 113, 11441–11446. http://www.pnas.org/lookup/doi/10.1073/pnas.1604850113, DOI: 10.1073/pnas.1604850113. .
[66] J. Faraone, N. Fraser, M. Blott, P.H. Leong, SYQ: Learning Symmetric Quantization for Efficient Deep Neural Networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
[67] E. Fiesler, A. Choudry, H.J. Caulfield, Weight discretization paradigm for optical neural networks, Optical Interconnections and Networks 1281 (1990) 164, https://doi.org/10.1117/12.20700.
[68] M. Figurnov, M.D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, R. Salakhutdinov, Spatially Adaptive Computation Time for Residual Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 1790–1799, https://doi.org/10.1109/CVPR.2017.194.
[69] FPGA, I., Intel FPGA Development Tools - Intel FPGA. https://www.intel.com/content/www/us/en/software/programmable/overview.html. .
[70] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: International Conference on Learning Representations (ICLR), 2019.
[71] K. Fukushima, Neocognitron: A hierarchical neural network capable of visual pattern recognition, Neural Networks 1 (1988) 119–130, https://doi.org/10.1016/0893-6080(88)90014-7.
[72] Gale, T., Elsen, E., Hooker, S., 2019. The State of Sparsity in Deep Neural Networks. ArXiv preprint http://arxiv.org/abs/1902.09574. .
[73] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, C.Z. Xu, L. Dudziak, R. Mullins, C.Z. Xu, Dynamic Channel Pruning: Feature Boosting and Suppression, in: International Conference on Learning Representations (ICLR), 2019, pp. 1–14, http://arxiv.org/abs/1810.05331.

[74] J. Glossner, P. Blinzer, J. Takala, HSA-enabled DSPs and accelerators, in: 2015 IEEE Global Conference on Signal and Information Processing GlobalSIP 2015, 2016, pp. 1407–1411, https://doi.org/10.1109/GlobalSIP.2015.7418430.

[75] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, J. Yan, Differentiable soft quantization: Bridging full-precision and low-bit neural networks, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019, pp. 4851–4860, https://doi.org/10.1109/ICCV.2019.00495.

[76] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing Deep Convolutional Networks using Vector Quantization, in: International Conference on Learning Representations(ICLR), 2014.

[77] Google, Hosted models — TensorFlow Lite. https://www.tensorflow.org/lite/guide/hosted_models. .

[78] Google, 2018. google/gemmlowp: Low-precision matrix multiplication. https://github.com/google/gemmlowp. https://github.com/google/gemmlowp. .

[79] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.J. Yang, E. Choi, MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. 1586–1595, https://doi.org/10.1109/CVPR.2018.00171.

[80] Gou, J., Yu, B., Maybank, S.J., Tao, D., 2020. Knowledge Distillation: A Survey. ArXiv preprint http://arxiv.org/abs/2006.05525. .

[81] Graham, B., 2017. Low-Precision Batch-Normalized Activations. ArXiv preprint, 1–16 http://arxiv.org/abs/1702.08231. .

[82] Graves, A., 2016. Adaptive Computation Time for Recurrent Neural Networks. ArXiv preprint, 1–19 http://arxiv.org/abs/1603.08983. .

[83] K. Greff, R.K. Srivastava, J. Schmidhuber, Highway and Residual Networks learn Unrolled Iterative Estimation, in: International Conference on Learning Representations(ICLR), 2016, pp. 1–14.

[84] Gudovskiy, D.A., Rigazio, L., 2017. ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks. ArXiv preprint http://arxiv.org/abs/1706.02393. .

[85] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, H. Yang, Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 37 (2018) 35–47, https://doi.org/10.1109/TCAD.2017.2705069, https://ieeexplore.ieee.org/abstract/document/7930521/.

[86] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, A Survey of FPGA-Based Neural Network Accelerator, in: ACM Transactions on Reconfigurable Technology and Systems 9, 2017.

[87] Guo, Y., 2018. A Survey on Methods and Theories of Quantized Neural Networks. ArXiv preprint http://arxiv.org/abs/1808.04752. .

[88] Y. Guo, A. Yao, Y. Chen, Dynamic Network Surgery for Efficient DNNs, in: Advances in Neural Information Processing Systems (NIPS), 2016, pp. 1379–1387, http://papers.nips.cc/paper/6165-dynamic-network-surgery-for-efficient-dnns.

[89] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: International Conference on Machine Learning (ICML), 2015, pp. 1737–1746.

[90] P. Gysel, J. Pimentel, M. Motamedi, S. Ghiasi, Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks, IEEE Transactions on Neural Networks and Learning Systems 29 (2018) 1–6, https://doi.org/10.1109/TNNLS.2018.2808319, https://ieeexplore.ieee.org/abstract/document/8318896/.

[91] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: Efficient Inference Engine on Compressed Deep Neural Network, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), IEEE, 2016, pp. 243–254, https://doi.org/10.1109/ISCA.2016.30.

[92] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, in: International Conference on Learning Representations(ICLR), 2016, pp. 199–203.

[93] Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., Elsen, E., Vajda, P., Paluri, M., Tran, J., Catanzaro, B., Dally, W.J., 2016c. DSD: Dense-Sparse-Dense Training for Deep Neural Networks, in: International Conference on Learning Representations(ICLR). http://arxiv.org/abs/1607.04381. .

[94] Han, S., Pool, J., Tran, J., Dally, W.J., 2015. Learning both Weights and Connections for Efficient Neural Networks, in: Advances in Neural Information Processing Systems (NIPS), pp. 1135–1143. http://arxiv.org/abs/1506.02626, DOI: 10.1016/S0140-6736(95)92525-2. .

[95] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., Ng, A.Y., 2014. Deep Speech: Scaling up end-to-end speech recognition. ArXiv preprint, 1–12 http://arxiv.org/abs/1412.5567. .

[96] HANSON, S., 1989. Comparing biases for minimal network construction with back-propagation, in: Advances in Neural Information Processing Systems (NIPS), pp. 177–185. .

[97] Hassibi, B., Stork, D.G., Wolff, G.J., 1993. Optimal brain surgeon and general network pruning. 10.1109/icnn.1993.298572. .

[98] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 171–180, https://doi.org/10.3389/fpsyg.2013.00124.

[99] He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y., 2018. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), International Joint Conferences on Artificial Intelligence Organization, California. pp. 2234–2240. http://arxiv.org/abs/1808.06866, 10.24963/ijcai.2018/309. .

[100] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[101] Y. He, X. Zhang, J. Sun, Channel Pruning for Accelerating Very Deep Neural Networks, in: IEEE International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 1398–1406, https://doi.org/10.1109/ICCV.2017.155.

[102] Hinton, G., 2012. Neural networks for machine learning. Technical Report. Coursera. .

[103] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. ArXiv preprint, 1–18http://arxiv.org/abs/1207.0580. .

[104] L. Hou, Q. Yao, J.T. Kwok, Loss-aware Binarization of Deep Networks, in: International Conference on Learning Representations(ICLR), 2017.

[105] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv preprint http://arxiv.org/abs/1704.04861. .

[106] Hu, H., Peng, R., Tai, Y.W., Tang, C.K., 2016. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. ArXiv preprint http://arxiv.org/abs/1607.03250. .

[107] Q. Hu, P. Wang, J. Cheng, From hashing to CNNs: Training binary weight networks via hashing, in: AAAI Conference on Artificial Intelligence, 2018, pp. 3247–3254.

[108] Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., Weinberger, K., 2018. Multi-scale dense networks for resource efficient image classification, in: International Conference on Learning Representations(ICLR). http://image-net.org/challenges/talks/. .

[109] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely Connected Convolutional Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 2261–2269. https://ieeexplore.ieee.org/document/8099726/, 10.1109/CVPR.2017.243. .

[110] Huang, G.B., Learned-miller, E., 2014. Labeled faces in the wild: Updates and new reporting procedures. Dept. Comput. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, Tech. Rep 14, 1–5. .

[111] Huang, Z., Wang, N., 2018. Data-Driven Sparse Structure Selection for Deep Neural Networks, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). volume 11220 LNCS, pp. 317–334. http://link.springer.com/10.1007/978-3-030-01270-0_19. .

[112] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized Neural Networks, Advances in Neural Information Processing Systems (NIPS) (2016) 4114–4122, http://papers.nips.cc/paper/6573-binarized-neural-networks.

[113] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations, Journal of Machine Learning Research 18 (18) (2016) 181–187, http://arxiv.org/abs/1609.07061.

[114] Hwang, K., Sung, W., 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and -1, in: 2014 IEEE Workshop on Signal Processing Systems (SiPS), IEEE. pp. 1–6. https://ieeexplore.ieee.org/abstract/document/6986082/, DOI: 10.1109/SiPS.2014.6986082. .

[115] Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, in: ArXiv e-prints. https://arxiv.org/abs/1602.07360 http://arxiv.org/abs/1602.07360, DOI: 10.1007/978-3-319-24553-9. .

[116] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, L. Van Gool, AI benchmark: All about deep learning on smartphones in 2019, in: Proceedings - 2019 International Conference on Computer Vision Workshop ICCVW 2019, 3617–3635, 2019, https://doi.org/10.1109/ICCVW.2019.00447.

[117] Imagination, PowerVR - embedded graphics processors powering iconic products. https://www.imgtec.com/graphics-processors/. .

[118] Intel, OpenVINO Toolkit. https://docs.openvinotoolkit.org/latest/index.html. .

[119] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, International Conference on Machine Learning (ICML) (2015) 448–456, http://arxiv.org/abs/1502.03167.

[120] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. 2704–2713, https://doi.org/10.1109/CVPR.2018.00286.

[121] Jia, Z., Tillman, B., Maggioni, M., Scarpazza, D.P., 2019. Dissecting the graphcore IPU architecture via microbenchmarking. ArXiv preprint. .

[122] Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, Li Fei-Fei, 2009. ImageNet: A large-scale hierarchical image database. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 248–25510.1109/cvprw.2009.5206848 .

[123] Jianchang Mao, Mohiuddin, K., Jain, A., 1994. Parsimonious network design and feature selection through node pruning, in: Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5), IEEE Comput. Soc. Press. pp. 622–624. http://ieeexplore.ieee.org/document/577060/, DOI: 10.1109/icpr.1994.577060. .

[124] Y. Jiao, L. Han, X. Long, Hanguang 800 NPU - The Ultimate AI Inference Solution for Data Centers, in: 2020 IEEE Hot Chips 32 Symposium (HCS), IEEE, 2020, pp. 1–29, https://doi.org/10.1109/HCS49909.2020.9220619.

[125] Jouppi, N.P., Borchers, A., Boyle, R., Cantin, P.l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Young, C., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C.R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Patil, N., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Patterson, D., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Agrawal, G., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Bajwa, R., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Bates, S., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H., Bhatia, S., Boden, N., 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. ACM SIGARCH Computer Architecture News 45, 1–12. http://dl.acm.org/citation.cfm?doid=3140659.3080246, 10.1145/3140659.3080246. .

[126] Judd, P., Delmas, A., Sharify, S., Moshovos, A., 2017. Cnvlutin2: Ineffectual-Activation-and-Weight-Free Deep Neural Network Computing. ArXiv preprint, 1–6 https://arxiv.org/abs/1705.00125. .

[127] Jung, S., Son, C., Lee, S., Son, J., Kwak, Y., Han, J.J., Hwang, S.J., Choi, C., 2018. Learning to Quantize Deep Networks by Optimizing Quantization Intervals with Task Loss. Revue Internationale de la Croix-Rouge et Bulletin international des Sociétés de la Croix-Rouge http://arxiv.org/abs/1808.05779, arXiv:1808.05779v2. .

[128] Kathail, V., 2020. Xilinx Vitis Unified Software Platform, in: Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, New York, NY, USA. pp. 173–174. https://dl.acm.org/doi/10.1145/3373087.3375887, DOI: 10.1145/3373087.3375887. .

[129] Keil, 2018. CMSIS NN Software Library. https://arm-software.github.io/CMSIS_5/NN/html/index.html. .

[130] Köster, U., Webb, T.J., Wang, X., Nassar, M., Bansal, A.K., Constable, W.H., Elibol, O.H., Gray, S., Hall, S., Hornof, L., Khosrowshahi, A., Kloss, C., Pai, R.J., Rao, N., 2017. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. ArXiv preprint http://arxiv.org/abs/1711.02213. .

[131] Krishnamoorthi, R., 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. ArXiv preprint 8, 667–668. http://cn.arxiv.org/pdf/1806.08342.pdf http://arxiv.org/abs/1806.08342, arXiv:1806.08342v1. .

[132] Krizhevsky, A., 2009. Learning Multiple Layers of Features from Tiny Images. Science Department, University of Toronto, Tech. 10.1.1.222.9220. .

[133] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks, in: Advances in Neural Information Processing Systems (NIPS), pp. 1–9. http://code.google.com/p/cuda-convnet/, doi: 10.1016/j.protcy.2014.09.007. .

[134] Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., Zinenko, O., 2020. MLIR: A Compiler Infrastructure for the End of Moore's Law. ArXiv preprint http://arxiv.org/abs/2002.11054. .

[135] Lavin, A., Gray, S., 2016. Fast Algorithms for Convolutional Neural Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 4013–4021. http://ieeexplore.ieee.org/document/7780804/ http://arxiv.org/abs/1312.5851, DOI: 10.1109/CVPR.2016.435. .

[136] Lebedev, V., Lempitsky, V., 2016. Fast ConvNets Using Group-Wise Brain Damage, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 2554–2564. http://openaccess.thecvf.com/content_cvpr_2016/html/Lebedev_Fast_ConvNets_Using_CVPR_2016_paper.html http://ieeexplore.ieee.org/document/7780649/, DOI: 10.1109/CVPR.2016.280. .

[137] Lebedev, V., Lempitsky, V., 2018. Speeding-up convolutional neural networks: A survey. BULLETIN OF THE POLISH ACADEMY OF SCIENCES TECHNICAL SCIENCES 66, 2018. http://www.czasopisma.pan.pl/Content/109869/PDF/05_799-810_00925_Bpast.No.66-6_31.12.18_K2.pdf?handler=pdf http://www.czasopisma.pan.pl/Content/109869/PDF/05_799-810_00925_Bpast.No.66-6_31.12.18_K2.pdf, 10.24425/bpas.2018.125927. .

[138] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444, https://doi.org/10.1038/nature14539.

[139] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (1998) 2278–2323, https://doi.org/10.1109/5.726791, http://ieeexplore.ieee.org/document/726791/.

[140] Y. LeCun, J.S. Denker, S.A. Solla, Optimal Brain Damage, Advances in Neural Information Processing Systems (NIPS) (1990) 598–605, https://doi.org/10.5555/109230.109298.

[141] Lee, N., Ajanthan, T., Torr, P.H., 2019. SnIP: Single-shot network pruning based on connection sensitivity, in: International Conference on Learning Representations(ICLR). .

[142] Lei, J., Gao, X., Song, J., Wang, X.L., Song, M.L., 2018. Survey of Deep Neural Network Model Compression. Ruan Jian Xue Bao/Journal of Software 29, 251–266. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049464636&doi=10.13328 10.13328/j.cnki.jos.005428. .

[143] W. Lei, H. Chen, Y. Wu, Compressing Deep Convolutional Networks Using K-means Based on Weights Distribution, in: Proceedings of the 2nd International Conference on Intelligent Information Processing, ACM Press, New York, New York, USA, 2017, pp. 1–6, https://doi.org/10.1145/3144789.3144803.

[144] C. Leng, H. Li, S. Zhu, R. Jin, Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM, in: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018.

[145] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, B. Dhoedt, The cascading neural network: building the Internet of Smart Things, Knowl. Inf. Syst. 52 (2017) 791–814, https://doi.org/10.1007/s10115-017-1029-1, http://link.springer.com/10.1007/s10115-017-1029-1.

[146] Li, F., Zhang, B., Liu, B., 2016. Ternary Weight Networks, in: Advances in Neural Information Processing Systems (NIPS). http://arxiv.org/abs/1605.04711. .

[147] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning Filters for Efficient ConvNets, in: International Conference on Learning Representations (ICLR), 2017, https://doi.org/10.1029/2009GL038531.

[148] H. Li, H. Zhang, X. Qi, Y. Ruigang, G. Huang, Improved Techniques for Training Adaptive Deep Networks, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2019, pp. 1891–1900, https://doi.org/10.1109/ICCV.2019.00198.

[149] Li, M., Liu, Y.I., Liu, X., Sun, Q., You, X.I.N., Yang, H., Luan, Z., Gan, L., Yang, G., Qian, D., 2020a. The Deep Learning Compiler: A Comprehensive Survey. ArXiv preprint 1, 1–36. http://arxiv.org/abs/2002.03794. .

[150] Y. Li, S. Gu, C. Mayer, L. Van Gool, R. Timofte, Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 8015–8024, https://doi.org/10.1109/CVPR42600.2020.00804.

[151] Z. Li, Y. Wang, T. Zhi, T. Chen, A survey of neural network accelerators, Front. Computer Sci. 11 (2017) 746–761, https://doi.org/10.1007/s11704-016-6159-1, http://link.springer.com/10.1007/s11704-016-6159-1.

[152] Z. Li, Y. Zhang, J. Wang, J. Lai, A survey of FPGA design for AI era, J. Semiconductors 41 (2020), https://doi.org/10.1088/1674-4926/41/2/021402.

[153] J. Lin, Y. Rao, J. Lu, J. Zhou, Runtime Neural Pruning, Advances in Neural Information Processing Systems (NIPS) (2017) 2178–2188, https://papers.nips.cc/paper/6813-runtime-neural-pruning.pdf.

[154] Lin, M., Chen, Q., Yan, S., 2014. Network in network, in: International Conference on Learning Representations(ICLR), pp. 1–10. .

[155] X. Lin, C. Zhao, W. Pan, Towards accurate binary convolutional neural network, Advances in Neural Information Processing Systems (NIPS) (2017) 345–353.

[156] Z. Lin, M. Courbariaux, R. Memisevic, Y. Bengio, Neural Networks with Few Multiplications, in: International Conference on Learning Representations (ICLR), 2016.

[157] J. Liu, P. Musialski, P. Wonka, J. Ye, Tensor Completion for Estimating Missing Values in Visual Data, IEEE Trans. Pattern Anal. Mach. Intell. 35 (2013) 208–220, https://doi.org/10.1109/TPAMI.2012.39, http://ieeexplore.ieee.org/document/6138863/.

[158] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning Efficient Convolutional Networks through Network Slimming, in: IEEE International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 2755–2763, https://doi.org/10.1109/ICCV.2017.298.

[159] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T.K.T. Cheng, J. Sun, MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning, in: IEEE International Conference on Computer Vision, 2019.

[160] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the Value of Network Pruning, International Conference on Learning Representations (ICLR) (2019) 1–11, http://arxiv.org/abs/1810.05270.

[161] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, K.T. Cheng, Bi-Real Net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11219 LNCS, 747–763, 2018, https://doi.org/10.1007/978-3-030-01267-0_44.

[162] Liu, Z.G., Mattina, M., 2019. Learning low-precision neural networks without Straight-Through Estimator (STE), in: IJCAI International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, California. pp. 3066–3072. https://www.ijcai.org/proceedings/2019/425, 10.24963/ijcai.2019/425. .

[163] J.H. Luo, J. Wu, AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference, Pattern Recogn. 107 (2020) , https://doi.org/10.1016/j.patcog.2020.107461, https://linkinghub.elsevier.com/retrieve/pii/S0031320320302648 107461.

[164] J.H.H. Luo, J. Wu, W. Lin, ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2017-Octob, 5068–5076, 2017, https://doi.org/10.1109/ICCV.2017.541.

[165] Y. Ma, N. Suda, Y. Cao, J.S. Seo, S. Vrudhula, Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA, in: FPL 2016–26th International Conference on Field-Programmable Logic and Applications, 2016, https://doi.org/10.1109/FPL.2016.7577356.

[166] O. Macchi, Coincidence Approach To Stochastic Point Process, Adv. Appl. Prob. 7 (1975) 83–122, https://doi.org/10.1017/s0001867800040313.

[167] Mariet, Z., Sra, S., 2016. Diversity Networks: Neural Network Compression Using Determinantal Point Processes, in: International Conference on Learning Representations(ICLR), pp. 1–13. http://arxiv.org/abs/1511.05077. .

[168] Mathieu, M., Henaff, M., LeCun, Y., 2013. Fast Training of Convolutional Networks through FFTs. ArXiv preprint http://arxiv.org/abs/1312.5851. .

[169] E. Medina, Habana Labs presentation. 2019 IEEE Hot Chips 31 Symposium, 2019.

[170] Mellempudi, N., Kundu, A., Mudigere, D., Das, D., Kaul, B., Dubey, P., 2017. Ternary Neural Networks with Fine-Grained Quantization. ArXiv preprint http://arxiv.org/abs/1705.01462. .
[171] Merolla, P., Appuswamy, R., Arthur, J., Esser, S.K., Modha, D., 2016. Deep neural networks are robust to weight binarization and other non-linear distortions. ArXiv preprint https://arxiv.org/abs/1606.01981 http://arxiv.org/abs/1606.01981. .
[172] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H., 2017. Mixed Precision Training, in: International Conference on Learning Representations(ICLR). http://arxiv.org/abs/1710.03740. .
[173] S. Migacz, 8-bit inference with TensorRT, GPU Technology Conference 2 (2017) 7, https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf.
[174] A. Mishra, E. Nurvitadhi, J.J. Cook, D. Marr, WRPN: Wide reduced-precision networks, in: International Conference on Learning Representations(ICLR), IEEE, 2018, pp. 1–11.
[175] Miyashita, D., Lee, E.H., Murmann, B., 2016. Convolutional Neural Networks using Logarithmic Data Representation. ArXiv preprint http://cn.arxiv.org/pdf/1603.01025.pdf http://arxiv.org/abs/1603.01025. .
[176] D. Molchanov, A. Ashukha, D. Vetrov, Variational dropout sparsifies deep neural networks, International Conference on Machine Learning (ICML) (2017) 3854–3863, https://dl.acm.org/citation.cfm?id=3305939.
[177] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning Convolutional Neural Networks for Resource Efficient Inference, International Conference on Learning Representations (ICLR) (2016) 1–17, http://arxiv.org/abs/1611.06440.
[178] Moss, D.J.M., Nurvitadhi, E., Sim, J., Mishra, A., Marr, D., Subhaschandra, S., Leong, P.H.W., 2017. High performance binary neural networks on the Xeon +FPGA platform, in: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE. pp. 1–4. https://ieeexplore.ieee.org/abstract/document/8056823/, 10.23919/FPL.2017.8056823. .
[179] M. Moudgill, J. Glossner, W. Huang, C. Tian, C. Xu, N. Yang, L. Wang, T. Liang, S. Shi, X. Zhang, D. Iancu, G. Nacer, K. Li, Heterogeneous Edge CNN Hardware Accelerator, in: The 12th International Conference on Wireless Communications and Signal Processing, IEEE, 2020, pp. 6–11.
[180] Muller, L.K., Indiveri, G., 2015. Rounding Methods for Neural Networks with Low Resolution Synaptic Weights. ArXiv preprint http://arxiv.org/abs/1504.05767. .
[181] R. Muthukrishnan, R. Rohini, LASSO: A feature selection technique in predictive modeling for machine learning, in: 2016 IEEE International Conference on Advances in Computer Applications (ICACA), IEEE, 2016, pp. 18–20, https://doi.org/10.1109/ICACA.2016.7887916.
[182] Neill, J.O., 2020. An Overview of Neural Network Compression. ArXiv preprint, 1–73 http://arxiv.org/abs/2006.03669. .
[183] NVIDIA Corporation, 2014. NVIDIA GeForce GTX 980 Featuring Maxwell, The Most Advanced GPU Ever Made. White Paper, 1–32 http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF. .
[184] NVIDIA Corporation, 2015. NVIDIA Tesla P100. White Paper https://www.nvidia.com/en-us/data-center/tesla-p100/. .
[185] NVIDIA Corporation, 2017a. NVIDIA DGX-1 With Tesla V100 System Architecture. White Paper http://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf. .
[186] NVIDIA Corporation, 2017b. NVIDIA Tesla V100 GPU Volta Architecture. White Paper, 53 http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf .
[187] NVIDIA Corporation, 2018a. NVIDIA A100 Tensor Core GPU. White Paper, 20–21. .
[188] NVIDIA Corporation, 2018b. NVIDIA Turing GPU Architecture. White Paper https://gpltech.com/wp-content/uploads/2018/11/NVIDIA-Turing-Architecture-Whitepaper.pdf. .
[189] Odena, A., Lawson, D., Olah, C., 2017. Changing Model Behavior at Test-Time Using Reinforcement Learning, in: International Conference on Learning Representations Workshops (ICLRW), International Conference on Learning Representations, ICLR. http://arxiv.org/abs/1702.07780. .
[190] ONNX, onnx/onnx: Open standard for machine learning interoperability. https://github.com/onnx/onnx. .
[191] Ouyang, J., Noh, M., Wang, Y., Qi, W., Ma, Y., Gu, C., Kim, S., Hong, K.i., Bae, W. K., Zhao, Z., Wang, J., Wu, P., Gong, X., Shi, J., Zhu, H., Du, X., 2020. Baidu Kunlun An AI processor for diversified workloads, in: 2020 IEEE Hot Chips 32 Symposium (HCS), IEEE. pp. 1–18. https://ieeexplore.ieee.org/document/9220641/, DOI: 10.1109/HCS49909.2020.9220641. .
[192] E. Park, J. Ahn, S. Yoo, Weighted-Entropy-Based Quantization for Deep Neural Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 7197–7205, https://doi.org/10.1109/CVPR.2017.761.
[193] Paszke, A., Gross, S., Bradbury, J., Lin, Z., Devito, Z., Massa, F., Steiner, B., Killeen, T., Yang, E., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. ArXiv preprint. .
[194] R. Pilipović, P. Bulić, V. Risojević, Compression of convolutional neural networks: A short survey, in: 2018 17th International Symposium on INFOTEH-JAHORINA INFOTEH 2018 - Proceedings, IEEE, 2018, pp. 1–6, https://doi.org/10.1109/INFOTEH.2018.8345545.
[195] A. Polyak, L. Wolf, Channel-level acceleration of deep face representations, IEEE Access 3 (2015) 2163–2175, https://doi.org/10.1109/ACCESS.2015.2494536, http://ieeexplore.ieee.org/document/7303876/.
[196] Preuser, T.B., Gambardella, G., Fraser, N., Blott, M., 2018. Inference of quantized neural networks on heterogeneous all-programmable devices, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE. pp. 833–838. http://ieeexplore.ieee.org/document/8342121/, 10.23919/DATE.2018.8342121. .
[197] Prost-Boucle, A., Bourge, A., Petrot, F., Alemdar, H., Caldwell, N., Leroy, V., 2017. Scalable high-performance architecture for convolutional ternary neural networks on FPGA, in: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE. pp. 1–7. https://hal.archives-ouvertes.fr/hal-01563763 http://ieeexplore.ieee.org/document/8056850/, 10.23919/FPL.2017.8056850. .
[198] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, N. Sebe, Binary neural networks: A survey, Pattern Recogn. 105 (2020) , https://doi.org/10.1016/j.patcog.2020.107281, https://linkinghub.elsevier.com/retrieve/pii/S0031320320300856 107281.
[199] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, J. Song, Forward and Backward Information Retention for Accurate Binary Neural Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 2247–2256, https://doi.org/10.1109/CVPR42600.2020.00232.
[200] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A., 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, in: European Conference on Computer Vision, Springer. pp. 525–542. http://arxiv.org/abs/1603.05279 http://link.springer.com/10.1007/978-3-319-46493-0_32, DOI: 10.1007/978-3-319-46493-0_32. .
[201] R. Reed, Pruning Algorithms - A Survey, IEEE Trans. Neural Networks 4 (1993) 740–747, https://doi.org/10.1109/72.248452, http://ieeexplore.ieee.org/document/248452/.
[202] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, J. Kepner, Survey and Benchmarking of Machine Learning Accelerators, in: 2019 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2019, pp. 1–9, https://doi.org/10.1109/HPEC.2019.8916327.
[203] Richard Chuang, Oliyide, O., Garrett, B., 2020. Introducing the Intel Vision Accelerator Design with Intel Arria 10 FPGA. White Paper. .
[204] A. Rodriguez, E. Segal, E. Meiri, E. Fomenko, Y.J. Kim, H. Shen, Lower Numerical Precision Deep Learning Inference and Training, Intel White Paper (2018) 1–19, https://software.intel.com/sites/default/files/managed/db/92/Lower-Numerical-Precision-Deep-Learning-Jan2018.pdf.
[205] Rotem, N., Fix, J., Abdulrasool, S., Catron, G., Deng, S., Dzhabarov, R., Gibson, N., Hegeman, J., Lele, M., Levenstein, R., Montgomery, J., Maher, B., Nadathur, S., Olesen, J., Park, J., Rakhov, A., Smelyanskiy, M., Wang, M., 2018. Glow: Graph lowering compiler techniques for neural networks. ArXiv preprint. .
[206] Ruffy, F., Chahal, K., 2019. The State of Knowledge Distillation for Classification. ArXiv preprint http://arxiv.org/abs/1912.10850. .
[207] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, Int. J. Comput. Vision 115 (2015) 211–252, https://doi.org/10.1007/s11263-015-0816-y, http://link.springer.com/10.1007/s11263-015-0816-y,.
[208] D. Saad, E. Marom, Training Feed Forward Nets with Binary Weights Via a Modified CHIR Algorithm, Complex Systems 4 (1990) 573–586, https://www.complex-systems.com/pdf/04-5-5.pdf.
[209] S. Sabour, N. Frosst, G.E. Hinton, Dynamic routing between capsules, Advances in Neural Information Processing Systems (NIPS) (2017) 3857–3867.
[210] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How does batch normalization help optimization?, Advances in Neural Information Processing Systems (NIPS) (2018) 2483–2493
[211] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, in: International Conference on Learning Representations(ICLR), 2013.
[212] Settle, S.O., Bollavaram, M., D'Alberto, P., Delaye, E., Fernandez, O., Fraser, N., Ng, A., Sirasao, A., Wu, M., 2018. Quantizing Convolutional Neural Networks for Low-Power High-Throughput Inference Engines. ArXiv preprint http://arxiv.org/abs/1805.07941. .
[213] M. Shen, K. Han, C. Xu, Y. Wang, Searching for accurate binary neural architectures, in: Proceedings - 2019 International Conference on Computer Vision Workshop ICCVW 2019, 2041–204410.1109/ICCVW.2019.00256, 2019.
[214] Shen, X., Yi, B., Zhang, Z., Shu, J., Liu, H., 2016. Automatic Recommendation Technology for Learning Resources with Convolutional Neural Network, in: Proceedings - 2016 International Symposium on Educational Technology, ISET 2016, pp. 30–34. DOI: 10.1109/ISET.2016.12. .
[215] Sheng, T., Feng, C., Zhuo, S., Zhang, X., Shen, L., Aleksic, M., 2018. A Quantization-Friendly Separable Convolution for MobileNets. 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), 14–18 https://ieeexplore.ieee.org/document/8524017/, DOI: 10.1109/EMC2.2018.00011. .
[216] T. Simons, D.J. Lee, A review of binarized neural networks, Electronics (Switzerland) 8 (2019), https://doi.org/10.3390/electronics8060661.
[217] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, in: International Conference on Learning Representations(ICLR), 2014, pp. 1–14.

[218] Singh, P., Kumar Verma, V., Rai, P., Namboodiri, V.P., 2019. Play and Prune: Adaptive Filter Pruning for Deep Model Compression, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, California. pp. 3460–3466. https://www.ijcai.org/proceedings/2019/480, 10.24963/ijcai.2019/480. .
[219] Society, I.C., Committee, M.S., 2008. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2008 2008, 1–70. 10.1109/IEEESTD.2008.4610935. .
[220] D. Soudry, I. Hubara, R. Meir, Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights, Advances in Neural Information Processing Systems (NIPS) (2014) 963–971, https://dl.acm.org/doi/abs/10.5555/2968826.2968934.
[221] Srinivas, S., Babu, R.V., 2015. Data-free parameter pruning for Deep Neural Networks, in: Procedings of the British Machine Vision Conference 2015, British Machine Vision Association. pp. 1–31. http://www.bmva.org/bmvc/2015/papers/paper031/index.html http://arxiv.org/abs/1507.06149, DOI: 10.5244/C.29.31. .
[222] Srivastava, N., Hinton, G.,..., A.K.T.j.o.m., 2014, U., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15, 1929–1958. http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign= buffer, 10.5555/2627435.2670313. .
[223] J. Sun, X. Luo, H. Gao, W. Wang, Y. Gao, X. Yang, Categorizing Malware via A Word2Vec-based Temporal Convolutional Network Scheme, Journal of Cloud Computing 9 (2020), https://doi.org/10.1186/s13677-020-00200-y.
[224] M. Sun, Z. Song, X. Jiang, J. Pan, Y. Pang, Learning Pooling for Convolutional Neural Network, Neurocomputing 224 (2017) 96–104, https://doi.org/10.1016/j.neucom.2016.10.049, DOI: 10.1016/j.neucom.2016.10.049.
[225] V. Sze, Y.H.H. Chen, T.J.J. Yang, J.S. Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, Proc. IEEE 105 (2017) 2295–2329, https://doi.org/10.1109/JPROC.2017.2761740, http://ieeexplore.ieee.org/document/8114708/.
[226] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2015, pp. 1–9, https://doi.org/10.1109/CVPR.2015.7298594.
[227] TansorFlow, Fixed Point Quantization. https://www.tensorflow.org/lite/guide. .
[228] Technologies, Q., 2019. Snapdragon Neural Processing Engine SDK. https://developer.qualcomm.com/docs/snpe/index.html .
[229] Tencent, 2019. NCNN is a high-performance neural network inference framework optimized for the mobile platform. https://github.com/Tencent/ncnn. .
[230] Tishbirani, R., 1996. Regression shrinkage and selection via the Lasso. https://statweb.stanford.edu/ tibs/lasso/lasso.pdf. .
[231] Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K., 2016. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17, 65–74 http://dl.acm.org/citation.cfm?doid=3020078.3021744, DOI: 10.1145/3020078.3021744. .
[232] H. Vanholder, Efficient Inference with TensorRT. Technical Report, 2016.
[233] Vanhoucke, V., Senior, A., Mao, M.Z., 2011. Improving the speed of neural networks on CPUs https://research.google/pubs/pub37631/. .
[234] S.I. Venieris, A. Kouris, C.S. Bouganis, Toolflows for Mapping Convolutional Neural Networks on FPGAs, ACM Comput. Surv. 51 (2018) 1–39, https://doi.org/10.1145/3186332, http://dl.acm.org/citation.cfm?doid=3212709.3186332.
[235] G. Venkatesh, E. Nurvitadhi, D. Marr, Accelerating Deep Convolutional Networks using low-precision and sparsity, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2017, pp. 2861–2865, https://doi.org/10.1109/ICASSP.2017.7952679.
[236] K. Wang, Z. Liu, Y. Lin, J. Lin, S. Han, HAQ: Hardware-Aware Automated Quantization With Mixed Precision, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 8604–8612, https://doi.org/10.1109/CVPR.2019.00881.
[237] N. Wang, J. Choi, D. Brand, C.Y. Chen, K. Gopalakrishnan, Training deep neural networks with 8-bit floating point numbers, Advances in Neural Information Processing Systems (NIPS) (2018) 7675–7684.
[238] Wang, P., Cheng, J., 2017. Fixed-Point Factorized Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 3966–3974. http://ieeexplore.ieee.org/document/8099905/, DOI: 10.1109/CVPR.2017.422. .
[239] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, J. Cheng, Two-Step Quantization for Low-bit Neural Networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4376–4384, https://doi.org/10.1109/CVPR.2018.00460.
[240] Z. Wang, J. Lu, C. Tao, J. Zhou, Q. Tian, Learning channel-wise interactions for binary convolutional neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 568–577, https://doi.org/10.1109/CVPR.2019.00066.
[241] Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H., 2016. Learning Structured Sparsity in Deep Neural Networks, in: Advances in Neural Information Processing Systems (NIPS), IEEE. pp. 2074–2082. https://dl.acm.org/doi/abs/10.5555/3157096.3157329, DOI: 10.1016/j.ccr.2008.06.009. .
[242] Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P., 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. ArXiv preprint, 1–20. .
[243] J. Wu, C. Leng, Y. Wang, Q. Hu, J. Cheng, Quantized Convolutional Neural Networks for Mobile Devices, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 4820–4828, https://doi.org/10.1109/CVPR.2016.521.
[244] S. Wu, G. Li, F. Chen, L. Shi, Training and Inference with Integers in Deep Neural Networks, in: International Conference on Learning Representations (ICLR), 2018.
[245] S. Wu, G. Li, L. Deng, L. Liu, D. Wu, Y. Xie, L. Shi, L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks, IEEE Transactions on Neural Networks and Learning Systems 30 (2019) 2043–2051, https://doi.org/10.1109/TNNLS.2018.2876179, https://ieeexplore.ieee.org/abstract/document/8528524/ https://ieeexplore.ieee.org/document/8528524/.
[246] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L.S. Davis, K. Grauman, R. Feris, BlockDrop: Dynamic Inference Paths in Residual Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. 8817–8826, https://doi.org/10.1109/CVPR.2018.00919.
[247] Xiaomi, 2019. MACE is a deep learning inference framework optimized for mobile heterogeneous computing platforms. https://github.com/XiaoMi/mace/. .
[248] Xilinx Inc, Accelerating DNNs with Xilinx Alveo Accelerator Cards (WP504), White Paper 504 (2018) 1–11, www.xilinx.com1.
[249] J. Xu, Y. Huan, L.R. Zheng, Z. Zou, A Low-Power Arithmetic Element for Multi-Base Logarithmic Computation on Deep Neural Networks, in: International System on Chip Conference IEEE, 2019, pp. 260–265, https://doi.org/10.1109/SOCC.2018.8618560.
[250] Xu, S., Huang, A., Chen, L., Zhang, B., 2020. Convolutional Neural Network Pruning: A Survey, in: 2020 39th Chinese Control Conference (CCC), IEEE. pp. 7458–7463. https://ieeexplore.ieee.org/document/9189610/, 10.23919/CCC50068.2020.9189610. .
[251] X. Xu, Q. Lu, L. Yang, S. Hu, D. Chen, Y. Hu, Y. Shi, Quantization of Fully Convolutional Networks for Accurate Biomedical Image Segmentation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8300–8308, https://doi.org/10.1109/CVPR.2018.00866.
[252] Z. Xu, Y.C. Hsu, J. Huang, Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks, in: International Conference on Learning Representations (ICLR)-Workshop, 2018.
[253] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, X.s. Hua, Quantization Networks, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 7300–7308, https://doi.org/10.1109/CVPR.2019.00748.
[254] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, G. Li, Training high-performance and large-scale deep neural networks with full 8-bit integers, Neural Networks 125 (2020) 70–82, https://doi.org/10.1016/j.neunet.2019.12.027.
[255] Ye, J., Lu, X., Lin, Z., Wang, J.Z., 2018. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. ArXiv preprint http://arxiv.org/abs/1802.00124. .
[256] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, J. Xin, Blended coarse gradient descent for full quantization of deep neural networks, Research in Mathematical Sciences 6 (2019), https://doi.org/10.1007/s40687-018-0177-6.
[257] D. Yogatama, G. Mann, in: S. Kaski, J. Corander (Eds.), Efficient Transfer Learning Method for Automatic Hyperparameter Tuning Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, 2014, pp. 1077–1085.
[258] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, S. Mahlke, Scalpel: Customizing DNN pruning to the underlying hardware parallelism, ACM SIGARCH Computer Architecture News 45 (2017) 548–560, https://doi.org/10.1145/3140659.3080215, http://dl.acm.org/citation.cfm?doid=3140659.3080215.
[259] J. Yu, L. Yang, N. Xu, J. Yang, T. Huang, Slimmable Neural Networks, in: International Conference on Learning Representations(ICLR), International Conference on Learning Representations ICLR, 2018, pp. 1–12.
[260] M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68 (2006) 49–67, https://doi.org/10.1111/j.1467-9868.2005.00532.x, http://doi.wiley.com/10.1111/j.1467-9868.2005.00532.x,.
[261] Z. Yuan, J. Hu, D. Wu, X. Ban, A dual-attention recurrent neural network method for deep cone thickener underflow concentration prediction, Sensors (Switzerland) 20 (2020) 1–18, https://doi.org/10.3390/s20051260.
[262] D. Zhang, J. Yang, D. Ye, G. Hua, LQ-Nets: Learned quantization for highly accurate and compact deep neural networks, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2018) 373–390, https://doi.org/10.1007/978-3-030-01237-3_23.
[263] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, B. Yu, Recent Advances in Convolutional Neural Network Acceleration, Neurocomputing 323 (2019) 37–51, https://doi.org/10.1016/j.neucom.2018.09.038, https://linkinghub.elsevier.com/retrieve/pii/S0925231218311007.

[264] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, Y. Chen, Cambricon-X: An accelerator for sparse neural networks, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE, 2016, pp. 1–12, URL:http://ieeexplore.ieee.org/document/7783723/.
[265] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. Salakhutdinov, Y. Bengio, Architectural complexity measures of recurrent neural networks, Advances in Neural Information Processing Systems (NIPS) (2016) 1830–1838.
[266] Zhang, Y., Zhao, C., Ni, B., Zhang, J., Deng, H., 2019b. Exploiting Channel Similarity for Accelerating Deep Convolutional Neural Networks. ArXiv preprint, 1–14 http://arxiv.org/abs/1908.02620. .
[267] R. Zhao, W. Song, W. Zhang, T. Xing, J.H. Lin, M. Srivastava, R. Gupta, Z. Zhang, Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17, ACM Press New York, New York, USA, 2017, pp. 15–24, https://doi.org/10.1145/3020078.3021741.
[268] Zhong, K., Zhao, T., Ning, X., Zeng, S., Guo, K., Wang, Y., Yang, H., 2020. Towards Lower Bit Multiplication for Convolutional Neural Network Training. ArXiv preprint http://arxiv.org/abs/2006.02804. .
[269] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights, in: International Conference on Learning Representations(ICLR), 2017.
[270] Zhou, H., Alvarez, J.M., Porikli, F., 2016a. Less Is More: Towards Compact CNNs, in: European Conference on Computer Vision, pp. 662–677. http://link.springer.com/10.1007/978-3-319-46493-0_40, DOI: 10.1007/978-3-319-46493-0_40. .
[271] S. Zhou, R. Kannan, V.K. Prasanna, Accelerating low rank matrix completion on FPGA, in: 2017 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2017, IEEE, 2018, pp. 1–7, https://doi.org/10.1109/RECONFIG.2017.8279771.
[272] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y., 2016b. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. ArXiv preprint abs/1606.0, 1–13. https://arxiv.org/abs/1606.06160. .
[273] S.C. Zhou, Y.Z. Wang, H. Wen, Q.Y. He, Y.H. Zou, Balanced Quantization: An Effective and Efficient Approach to Quantized Neural Networks, Journal of Computer Science and Technology 32 (2017) 667–682, https://doi.org/10.1007/s11390-017-1750-y.
[274] C. Zhu, S. Han, H. Mao, W.J. Dally, Trained Ternary Quantization, International Conference on Learning Representations (ICLR) (2017) 1–10, http://arxiv.org/abs/1612.01064.
[275] Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., Yan, J., Towards Unified INT8 Training for Convolutional Neural Network, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). http://arxiv.org/abs/1912.12607. .
[276] B. Zhuang, C. Shen, M. Tan, L. Liu, I. Reid, Structured binary neural networks for accurate image classification and semantic segmentation, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2019-June, 413–422, 2019, https://doi.org/10.1109/CVPR.2019.00050.
[277] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning Transferable Architectures for Scalable Image Recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 8697–8710.

Tailin Liang received the B.E. degree in Computer Science and B.B.A from the University of Science and Technology Beijing in 2017. He is currently working toward a Ph.D. degree in Computer Science at the School of Computer and Communication Engineering, University of Science and Technology Beijing. His current research interests include deep learning domain-specific processors and co-designed optimization algorithms.

John Glossner received the Ph.D. degree in Electrical Engineering from TU Delft in 2001. He is the Director of the Computer Architecture, Heterogeneous Computing, and AI Lab at the University of Science and Technology Beijing. He is also the CEO of Optimum Semiconductor Technologies and President of both the Heterogeneous System Architecture Foundation and Wireless Innovation Forum. John's research interests include the design of heterogeneous computing systems, computer architecture, embedded systems, digital signal processors, software defined radios, artificial intelligence algorithms, and machine learning systems.

Lei Wang received the B.E. and Ph.D. degrees in 2006 and 2012 from the University of Science and Technology Beijing. He then served as an assistant researcher at the Institute of Automation of the Chinese Academy of Sciences during 2012-2015. He was a joint Ph.D. of Electronic Engineering at The University of Texas at Dallas during 2009-2011. Currently, he is an adjunct professor at the School of Computer and Communication Engineering, University of Science and Technology Beijing.

Shaobo Shi received the B.E. and Ph.D. degrees in 2008 and 2014 from the University of Science and Technology Beijing. He then served as an assistant researcher at the Institute of Automation of the Chinese Academy of Sciences during 2014-2017. Currently, he is a deep learning domain-specific processor engineer at Huaxia General Processor Technology. As well serve as an adjunct professor at the School of Computer and Communication Engineering, University of Science and Technology Beijing.

Xiaotong Zhang received the M.E. and Ph.D. degrees from the University of Science and Technology Beijing in 1997 and 2000, respectively, where he was a professor of Computer Science and Technology. His research interest includes the quality of wireless channels and networks, wireless sensor networks,