COMPUTER RESEARCH AND MODELING
2024 VOL. 16 NO. 7 P. 1601–1619
DOI: 10.20537/2076-7633-2024-16-7-1601-1619

Ки&М

SPECIAL ISSUE

UDC: 004.8

Review of algorithmic solutions for deployment of neural networks on lite devices

S. A. Khan[a], S. Shulepina[b], D. Shulepin[c], R. A. Lukmanov[d]

Innopolis University,
1 Universitetskaya st., Innopolis, 420500, Russia

E-mail: [a] sameedkhandurrani@gmail.com, [b] sofi1221@mail.ru, [c] dshulepin2013@gmail.com,
[d] r.lukmanov@innopolis.ru

Received 27.10.2024, after completion — 16.11.2024
Accepted for publication 25.11.2024

In today's technology-driven world, lite devices like Internet of Things (IoT) devices and microcontrollers (MCUs) are becoming increasingly common. These devices are more energy-efficient and affordable, often with reduced features compared to the standard versions such as very limited memory and processing power for typical machine learning models. However, modern machine learning models can have millions of parameters, resulting in a large memory footprint. This complexity not only makes it difficult to deploy these large models on resource constrained devices but also increases the risk of latency and inefficiency in processing, which is crucial in some cases where real-time responses are required such as autonomous driving and medical diagnostics. In recent years, neural networks have seen significant advancements in model optimization techniques that help deployment and inference on these small devices. This narrative review offers a thorough examination of the progression and latest developments in neural network optimization, focusing on key areas such as quantization, pruning, knowledge distillation, and neural architecture search. It examines how these algorithmic solutions have progressed and how new approaches have improved upon the existing techniques making neural networks more efficient. This review is designed for machine learning researchers, practitioners, and engineers who may be unfamiliar with these methods but wish to explore the available techniques. It highlights ongoing research in optimizing networks for achieving better performance, lowering energy consumption, and enabling faster training times, all of which play an important role in the continued scalability of neural networks. Additionally, it identifies gaps in current research and provides a foundation for future studies, aiming to enhance the applicability and effectiveness of existing optimization strategies.

Keywords: quantization, neural architecture search, knowledge distillation, pruning, reinforcement learning, model compression

Citation: *Computer Research and Modeling*, 2024, vol. 16, no. 7, pp. 1601–1619.

КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2024 Т. 16 № 7 С. 1601–1619
DOI: 10.20537/2076-7633-2024-16-7-1601-1619

Ки&М

СПЕЦИАЛЬНЫЙ ВЫПУСК

УДК: 004.8

Обзор алгоритмических решений для развертывания нейронных сетей на легких устройствах

С. А. Кхан[a], С. Шулепина[b], Д. Шулепин[c], Р. А. Лукманов[d]

Университет Иннополис,
Россия, 420500, г. Иннополис, ул. Университетская, д. 1

E-mail: [a] sameedkhandurrani@gmail.com, [b] sofi1221@mail.ru, [c] dshulepin2013@gmail.com,
[d] r.lukmanov@innopolis.ru

Получено 27.10.2024, после доработки — 16.11.2024
Принято к публикации 25.11.2024

В современном мире, ориентированном на технологии, легкие устройства, такие как устройства Интернета вещей (IoT) и микроконтроллеры (MCU), становятся все более распространенными. Эти устройства более энергоэффективны и доступны по цене, но часто обладают урезанными возможностями, по сравнению со стандартными версиями, такими как ограниченная память и вычислительная мощность. Современные модели машинного обучения могут содержать миллионы параметров, что приводит к значительному росту требований по объему памяти. Эта сложность не только затрудняет развертывание больших моделей на устройствах с ограниченными ресурсами, но и увеличивает риск задержек и неэффективности при обработке данных, что критично в случаях, когда требуются ответы в реальном времени, таких как автономное вождение или медицинская диагностика.

В последние годы нейронные сети достигли значительного прогресса в методах оптимизации моделей, что помогает в развертывании и инференсе на этих небольших устройствах. Данный обзор представляет собой подробное исследование прогресса и последних достижений в оптимизации нейронных сетей, сосредотачиваясь на ключевых областях, таких как квантизация, прореживание, дистилляция знаний и поиск архитектур нейронных сетей. Обзор рассматривает, как эти алгоритмические решения развивались и как новые подходы улучшили существующие методы, делая нейронные сети более эффективными. Статья предназначена для исследователей, практиков и инженеров в области машинного обучения, которые могут быть незнакомы с этими методами, но хотят изучить доступные техники. В работе подчеркиваются текущие исследования в области оптимизации нейронных сетей для достижения лучшей производительности, снижения потребления энергии и ускорения времени обучения, что играет важную роль в дальнейшей масштабируемости нейронных сетей. Кроме того, в обзоре определяются пробелы в текущих исследованиях и закладывается основа для будущих исследований, направленных на повышение применимости и эффективности существующих стратегий оптимизации.

Ключевые слова: квантизация, поиск архитектуры нейронной сети, дистилляция знаний, обрезка, обучение с подкреплением, сжатие модели

Introduction

Today complex machine learning models are running directly or indirectly behind many tasks that have become a part of our daily lives. Most of these models are running on various lite devices such as mobile phones, microcontrollers, and Internet of Things (IoT) devices and can be found in various industries from healthcare and autonomous systems to finance and telecommunications. Usually these devices are limited in terms of memory and processing capabilities. Modern day machine learning models require a lot of energy consumption. For example, a study suggests that GPT-3 needs to "drink" around a 500 ml bottle of water for 10–50 responses [Li et al., 2023]. Such limitations make it difficult to deploy advanced AI algorithms on them and expensive to use them.

Deep learning models are designed to help devices make decisions or predictions by analyzing data. These models can be very large and require enormous computational resources to operate effectively. For instance, a model that can recognize objects in images might need extensive memory and processing power, which are often unavailable in small devices. This mismatch between the needs of complex models and the capabilities of tiny devices poses a significant challenge.

To address these issues, researchers have developed such techniques as quantization [Jacob et al., 2017], pruning [Han et al., 2015], knowledge distillation [Hinton, Vinyals, Dean, 2015], and neural architecture search [Elsken, Metzen, Hutter, 2019] to adapt deep learning models for use on small devices. Quantization slightly reduces the precision of model computation, thus saving memory and computational power. Pruning involves removing insignificant parts of the model, making it sparse and less complex. Neural Architecture Search optimizes the design of the model to improve its efficiency. Knowledge Distillation uses a more complex model (teacher model) to train a simpler model (student model) by simulating its behavior [Lin et al., 2023]. These algorithmic solutions help to make complex models more feasible for deployment on resource-constrained devices. We can find these solutions being utilized in machine learning models powering self-driving cars, wearable health devices, and various mobile applications.

This narrative review explores these techniques and how they have evolved. It provides an overview of the current methods used to adapt neural networks for small devices and discusses their development. The review is intended for machine learning engineers and enthusiasts who are curious about the latest solutions for running AI on limited-resource devices. Understanding these advancements might help readers get an insight into the progress being made in this field and identify areas for improvement.

Literature review

In recent years, the development of energy-efficient and highly-integrated platforms has paved the way for next-generation industrial artificial intelligence (AI) applications. These applications incorporate various sensors, processors, and functional modules, necessitating the adaptive transformation of diverse data representation formats. To achieve this, model compression and optimization techniques play a crucial role. This review examine multiple different approaches in the context of industrial IoT applications.

Quantization

The main idea of quantization is to reduce the model weights, which are usually stored as 32-bit floating-point numbers. According to Mark Horowitz [Horowitz, 2014] a significant drop in energy consumption during multiplication or addition operations occurs by reducing the number of bits. Thus, 32-bit floating-point addition is 30 times more costly than 8-bit integer addition and 32-bit floating-point multiplication is around 18 times more costly than 8-bit integer multiplication. Figure 1 represents the comparison of addition and multiplication operations cost on a 45 nm technology node of

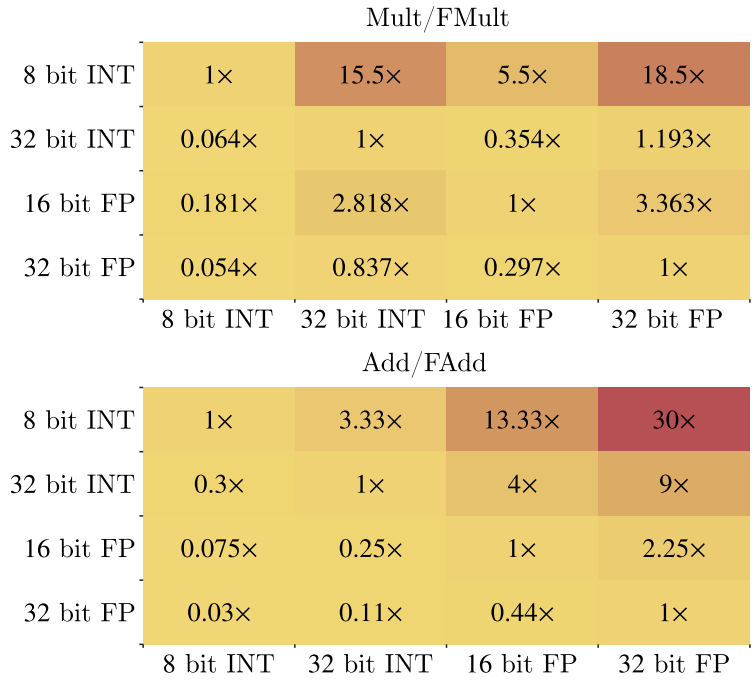

Figure 1. Comparison of rough energy cost for addition and multiplication operations

semiconductor manufacturing process, where the energy consumption for these operations is measured at an operating voltage typically around 0.9 V.

K-means based quantization

Han et al. introduced the *k*-means quantization technique [Han, Mao, Dally, 2016]. An original weight matrix of M 32-bit float values is clustered using a matrix of N-bit indexes of codebook, and a codebook containing cluster centroids. Each cluster index value is an index of codebook where the cluster centroid of that particular weight is stored. The weights may be reconstructed by replacing the centroid value in each cell of the matrix.

For example, the matrix of 16 32-bit float values may be represented as 16 2-bit integer indexes for codebook of 4 32-bit float centroids (Fig. 2). The size of the initial matrix is reduced by 3.2 times.

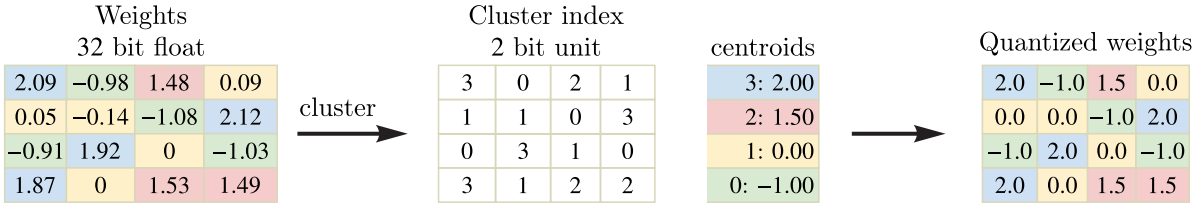

Figure 2. K-means weight quantization

However, the reconstruction error can still be minimized by grouping the gradients of each cluster, summing them and subtracting from the centroids. Those fine-tuned centroids result in more accurate weight matrix, as shown in Fig. 3.

Linear quantization

Jacob et al. provides a quantization scheme that was adopted in TensorFlow Lite, a version of TensorFlow for small and mobile devices [Jacob et al., 2017]. This schema use an affine mapping of integers to real numbers:

$$r = S(q - Z), \tag{1}$$

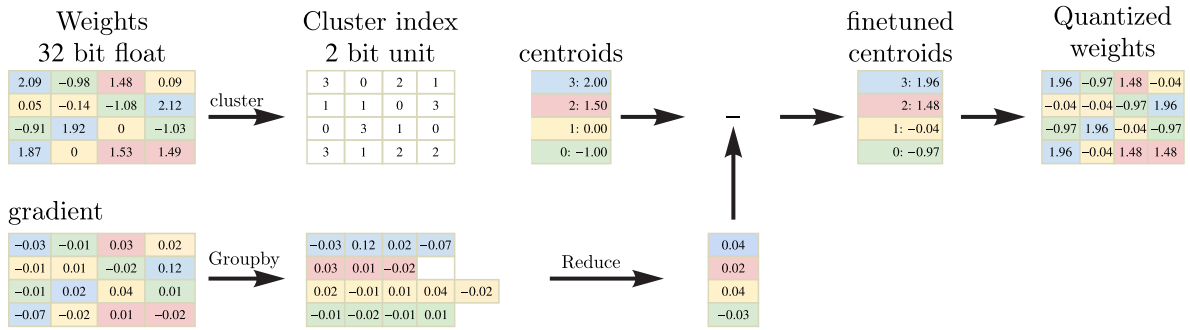

Figure 3. Fine-tuning quantized weights

where r is floating point weights, S is scale, floating point quantization parameter, q is quantized weights of signed integer values, Z is the zero-point, quantized integer representing $r = 0$.

Figure 4 shows a mapping from real values to quantized values.

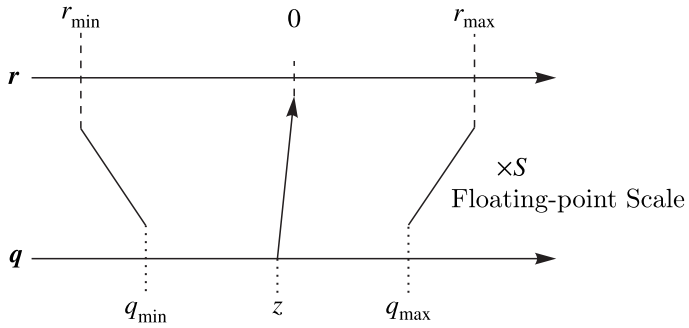

Figure 4. Linear quantization representation

The maximum and minimum values in these ranges help define how the real number scale is compressed into the integer scale. Since the maximum and minimum values on real scale and quantization scale are known before performing quantization, the quantization parameter and zero point can be obtained from the following equations:

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}. \tag{2}$$

The zero-point Z is found by calculating where the real number 0 fits on the integer scale:

$$Z = \text{round}\left(q_{min} - \frac{r_{min}}{S}\right). \tag{3}$$

Per channel quantization

Krishnamoorthi suggests using per-channel quantization technique in which the absolute maximum from each channel is stored and then used for quantization in each channel [Krishnamoorthi, 2018]:

$$q_x = \text{round}\left(\frac{r}{S}\right) + Z. \tag{4}$$

Each channel has a scale and zero-point based on the minimum and maximum values within that channel. It means the same floating-point value might be represented differently in different channels, depending on the specific quantization parameters for each channel. The overhead in this method arises when there is a need to store a lot of scales, which often utilize 32 bits, for every channel. Thus, the

method requires a significant amount of storage when dealing with a large number of channels, like in large-language models where thousands of channels are present. While per-channel quantization of weights is becoming more widely adopted, not all commercial hardware is capable of supporting it.

Per tensor quantization

Another similar approach is per-tensor [Nagel et al., 2019] quantization, in which a single scale is used for the entire tensor. This method is easy to implement and less computational and memory intensive as compared to per-channel quantization since it does not require scale and offset values for each channel but it can decrease the accuracy of the models, especially when output channels have different ranges.

Per vector quantization

VS-Quant [Dai et al., 2021] is a more granular per-vector scaled quantization technique. The channel is divided into vectors and a scaling factor is determined for each vector. This method is more suitable for models with high variability within channels. The effective bit width in VS-quant is slightly higher due to more vector scales, and can be calculated by:

$$\text{Effective bit width} = \text{Bits per element} + \frac{\text{Bits for scale}}{\text{Number of elements in vector}}. \quad (5)$$

Block data representation with shared microexponents (BDR-MX)

Microsoft introduced the Block Data Representations (BDR) framework which outlines methods for quantizing and scaling tensor values together. Further, a new numerical approach to quantization formats based on the MX datatype was introduced [Rouhani et al., 2023]. Among MX datatypes, MX4, MX6 and MX9 are the optimal choices because they effectively balance the trade-offs between precision, efficiency, hardware implementation complexity and application suitability. MX uses shared microexponents for scaling, a single bit for an exponent is shared between two elements, allowing these elements to use the same scaling factor which is applied at the hardware level. The MX datatype is being used in various applications including large language models (LLMs).

Dynamic range clipping for quantization

Capturing the range of neural network activations is necessary in maintaining higher accuracy of the model.

One of the approach is to track minimum and maximum values of activation during training, and then use Exponential Moving Averages (EMA) [Jacob et al., 2017] to smoothen any fluctuations in the values.

Running calibration batches of sample data is a widely used method [Verma et al., 2021] where sample data are run on the trained FP32 model and then statistical methods are applied to get an estimate of the range. For data with known distributions like Gaussian or Laplacian, we can do analytical clipping [Banner et al., 2018], otherwise, Kullback – Leibler Divergence can be considered to find values with minimal information loss.

OCTAV algorithm was also introduced [Sakr et al., 2022] to find the optimal limit for the quantized values. It uses an optimization approach, based on the Newton – Raphson method, to iteratively determine the optimal clipping scalar by reducing the MSE between the original values and quantized values.

Quantization-aware training

Quantization-Aware training (QAT) is a method in which model is trained initially on full-precision weights. Then the quantization is simulated on the high-precision weights and the network performs forward pass using the quantized weights. During the backward pass, gradients are calculated and used to update the original high-precision weights, not the quantized ones. This process is repeated

several times and the model eventually learns to make accurate predictions even when the weights are quantized. Normally for the backward pass, the gradients are calculated as

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Q(W)} \cdot \frac{\partial Q(W)}{\partial W},$$

where L is the loss function, W are the weights, and $Q(W)$ are the quantized weights.

But since quantization is discrete, the derivative of quantized weights over original weights would be

$$\frac{\partial Q(W)}{\partial W} = 0.$$

To cater this, QAT uses Straight-Through Estimator (STE) [Bengio, Léonard, Courville, 2013] which approximates that the quantization didn't happen and passes gradients through as if the weights were not quantized.

With STE, $\frac{\partial Q(W)}{\partial W}$ is approximated as the identity function $I(\cdot)$ and the equation becomes

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Q(W)}.$$

This allows the gradients to flow through quantized weights during training process, ensuring that weights are updated the model learns and improves its performance.

Memory-driven mixed low precision quantization

A memory aware quantization approach was introduced by Rusci et al. [Rusci, Capotondi, Benini, 2019]. It works by first analyzing the available memory which include the Read-Only memory (RO) and Write-Only memory (RW) of the machine and establishing the allowable maximum Read-Only memory (MRO) and maximum Write-Only memory (MRW) that can be used during inference. Based on the model, it initially set the bit precision to 8-bit and then it decides on possible bit-width for each layer from 2-bit, 4-bit and 8-bit precision. It iteratively performs forward pass, calculates the memory for input and output tensors, and if memory exceeds the MRO, it reduces the bit precision of the output tensor of the current layer and the input tensor of the next layer. Similarly, it performs backward pass and if memory exceeds the MRO, it reduces the bit precision of the input tensor of current layer and the output tensor of the previous layer. After making sure that total memory used for activate tensors and parameters, the quantized model is converted to integer only model by inserting Integer Channel-Normalization (ICN) layers. This is done to ensure that model runs effectively by performing integer arithmetic which is preferable for microcontrollers.

Table 1 shows the comparison between different quantization approaches including the memory driven mixed low precision technique on Integer-Only MobilenetV1 224 1.0. In comparison table, PL Quantization stands for per-layer quantization, PC Quantization stands for per-channel quantization, FB stands for folding of batch-norm parameters into weights and ICN stands for Integer Channel-Normalization.

Table 1. Memory-driven Quantization on Integer-Only MobilenetV1 224 1.0

Quantization method	Top1 accuracy	Weight memory footprint
Full-precision	70.90 %	16.27 MB
PL quantization + FB INT8	70.10 %	4.06 MB
PL quantization + ICN INT4	61.75 %	2.10 MB
PC quantization + ICN INT4	66.41 %	2.12 MB

While targeting specific hardware like ARM based microcontrollers, CMix-NN can be utilised [Capotondi et al., 2020]. CMix-NN is specialised for Cortex-M architectures, exploiting

the specific features of ARM's instruction set, such as vector arithmetic extensions to maximize computational efficiency while ensuring low memory usage.

Outlier free quantization for transformers

Transformers are specific kinds of attention models and have an attention head that requires quantization as well when quantizing a transformer model [Vaswani, 2017]. Sometimes quantization of transformers leads to outlier values in certain parts of the model causing accuracy loss. The outliers mainly appear in softmax function of self attention layer when the model tries not to update certain parts of the data by making attention update as close to zero as possible. To achieve this, the input to softmax is pushed to extremely high or extremely low, getting very small numbers as a result from softmax. The problem occurs when these extreme values become outliers during quantization as these extreme values cannot be represented well in low-bit formats like INT8, causing accuracy problems. To overcome this problem, two solutions were suggested [Bondarenko, Nagel, Blankevoort, 2023] namely clipped softmax and gated attention. Clipped softmax introduces a stretch factor ζ and a shift factor γ and using following equation it adjusts the values so that they remain within a range of 0 and 1, without the outliers:

$$\text{Clipped Softmax}(x;\ \zeta,\ \gamma) = \text{clip}((\zeta - \gamma) \cdot \text{softmax}(x) + \gamma,\ 0,\ 1). \tag{6}$$

In gated attention mechanism, a gating function is added to the standard attention. This gating function $G(X)$ is a lightweight neural network that is trained jointly with the model. The standard attention function is replaced with updated gated attention function in each layer of the transformer:

$$\text{Gated Attention}(x) = \text{sigmoid}(G(x)) \odot \text{softmax}(\textit{Attention scores}). \tag{7}$$

These solutions helps avoid the outliers and provides quantization of transformers with improved accuracy.

Pruning

In 2005, Drachman [Drachman, 2005] performed Stereologic studies which shows that an adult human brain has average 7000 synaptic connections on average and undergoes structural brain changes. According to the study, losses in less critical synaptic connections may not significantly affect overall brain function.

A similar study was done by Peter Huttenlocher who did a groundbreaking research on synaptic development [Walsh, 2013]. Peter demonstrated that billions of synapses in the human cerebral cortex are formed in infancy, which are then pruned during early childhood as shown in Fig. 5 [Huttenlocher, 1990].

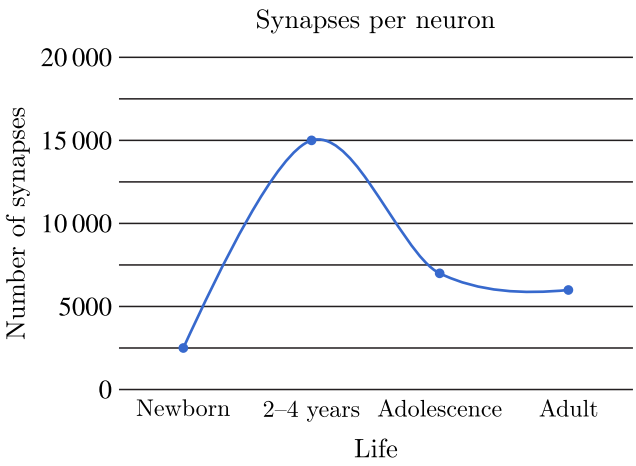

Figure 5. Average synapses per neuron

Pruning in neural networks is a similar approach which was first introduced by LeCun et al. in 1989 under Optimal Brain Damage [LeCun, Denker, Solla, 1989]. In Optimal Brain Damage, unimportant weights are removed and neural network speed is significantly improved while the accuracy is also slightly improved. This confirms the usefulness of the pruning methods in a real-world application and their prevalence in nature. Interestingly, pruning is a must in biological brains.

Magnitude based pruning

A three-step method was developed [Han et al., 2015] to prune redundant connections. The steps include network training in which the connection importance is learnt, then comes the pruning step where the weights below a certain threshold are removed, and lastly retraining the network to find out the weights for remaining connections of sparse neural network as illustrated in Fig. 6. The retraining of the network allows to remove the neurons without any input or output connections as shown in Fig. 7. This approach eventually reduces the parameters of AlexNet by 9x and VGG-16 by 13x without loss of accuracy.

Cyclical pruning

One of the drawbacks of magnitude based pruning is that the removed weights cannot be recovered. In some cases, it is important to recover weights for example when important weights are removed mistakenly during pruning and when some specific weight configurations result in better accuracy. Cyclic pruning follows a periodic schedule. If weights are pruned in a cycle, they can be potentially recovered in a next cycle. The possibility to recover weights helps recover important weights and make neural networks more efficient. The Table 2 shows accuracy of various models on CIFAR-10 dataset [Krizhevsky, Hinton, 2009] with 90 % pruning ratio. The magnitude based pruning was run for 100 epochs, and cyclical pruning was run for 5 cycles of 20 epochs each. Similarly, Table 3 shows accuracy of various models on Imagenet dataset [Deng et al., 2009] with 70 % pruning ratio. The magnitude based pruning was run for 60 epochs, and cyclical pruning was run for 3 cycles of 20 epochs each [Srinivas et al., 2022].

Table 2. Accuracy on CIFAR10 dataset — magnitude based pruning and cyclical pruning

Model	Magnitude based pruning	Cyclic pruning
ResNet56	92.35 % ± 0.1	92.41 % ± 0.1
Mobilenet	84.99 % ± 0.3	86.99 % ± 0.3

Table 3. Accuracy on Imagenet dataset — magnitude based pruning and cyclical pruning

Model	Magnitude based pruning	Cyclic pruning
ResNet18	69.2 %	69.4 %
ResNet50	75.9 %	75.7 %
EfficientNet	68 %	69.9 %
MobilenetV2	61.3 %	64.4 %

Scaling based pruning

Liu et al. introduced scaling based pruning [Liu et al., 2017]. A scaling factor is associated with each filter in convolutional layers. The scaling factor refers to a parameter that determines how aggressively or conservatively weights are pruned. If the scaling factor is quite small for a channel, the channel is then pruned to reduce the size of the model.

Structured pruning

To further improve the deployment of neural networks on resource constrained devices Wen et al. proposed a technique called Structured Sparsity Learning (SSL) [Wen et al., 2016]. The technique

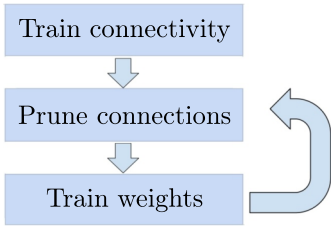

Figure 6. Steps involved in pruning method by Han S. et al.

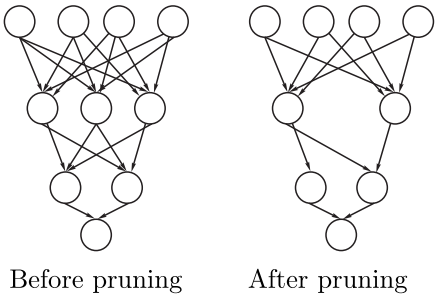

Figure 7. DNN before and after pruning

aims at regularizing the structures of deep neural networks by removing entire rows or columns in the weight matrices. As compared to three-step pruning approach by Han et al. which gives 42.80 % top-1 error rate on AlexNet on ILSVRC 2012, SSL provides a slightly better accuracy with 42.53 % top-1 error rate. However, the major improvement is in performance. On three-step pruning the speedup on CPUs ranges from 1.27× to 1.68× and on GPUs from 1.25× to 1.72×, while SSL consistently achieves better speedups, averaging around 5.1× on CPUs and 3.1× on GPUs.

Mao et al. thoroughly explored the effects of different sparsity techniques on neural networks [Mao et al., 2017]. The sparsity techniques were divided into four major categories based on their dimensions. Fine-grained Sparsity with 0 dimension, Vector-level Sparsity with 1 dimension, Kernel-level Sparsity with 2 dimensions and Filter-level Sparsity with 3 dimensions as shown in Fig. 8. It was found that neural networks with structured pruning are easy to accelerate as compared to those with unstructured pruning. The main reason behind that is structured pruning removes the extra columns of the matrix making the multiplication process faster. However, the fine-grained sparsity gives the highest accuracy on multiple models with same density. The comparison is represented in Table 4.

Table 4. Top-5 accuracies on various granularity levels

Model	Density	Kernel level Acc	Vector level Acc	Fine-grained Acc
AlexNet	24.80 %	79.20 %	79.94 %	80.41 %
VGG-16	23.50 %	89.70 %	90.48 %	90.56 %
GoogLeNet	38.40 %	88.83 %	89.11 %	89.40 %
ResNet-50	40 %	92.07 %	92.26 %	92.34 %
DenseNet-121	30.10 %	91.56 %	91.89 %	92.21 %

NVIDIA came up with a compression technique in 2021 for structured pruning which allows one to keep 2-bits Indices as well. It is supported by NVIDIA's Ampere GPU architecture, which delivers up to 2x acceleration.

Average percentage of zero based pruning

Hu et al. introduced another approach for pruning which is based on mainly two steps [Hu et al., 2016]. The first one is to apply ReLU to generate zeros in the output activation and second one is

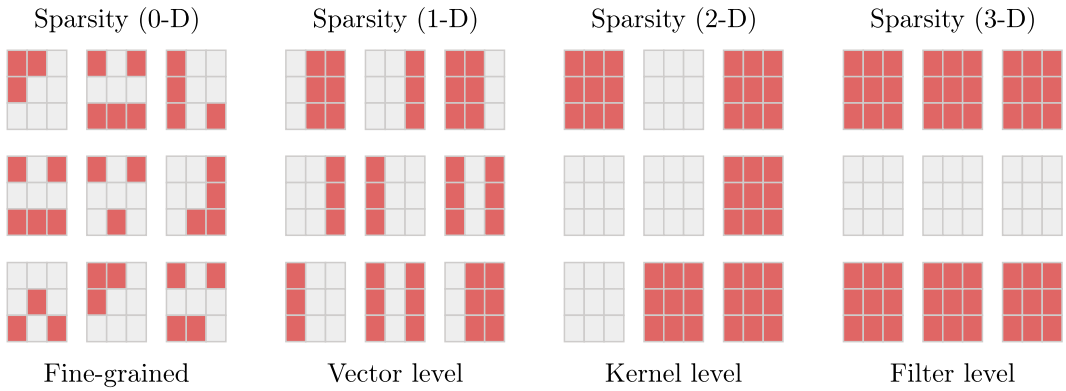

Figure 8. Different structure of sparsity in a 4-dimensional weight tensor

calculating Average Percentage of Zero activations (APoZ) to measure how important each neuron is. After these steps, the channel with highest APoZ can be pruned. The approach suggested by Han et al. [Han et al., 2015] prunes neurons by removing all their connections. This is ineffective espacially when there are many connections. APoZ based pruning uses APoZ metric to identify neurons that do not contribute much and directly prunes them.

Regression based pruning

He et al. proposed a technique by combining regression based channel selection and least square reconstruction [He, Zhang, Sun, 2017]. It largely minimizes the reconstruction error of corresponding layer's outputs. One drawback of this technique is that it works with each individual layer, and does not provide full picture of complete neural network.

NetAdapt

NetAdapt is an iterative method to finetune the model [Yang et al., 2018]. In each iteration, the latency is reduced by a certain amount which is manually defined. For each layer of neural network, the layer is pruned such that the latency reduction meets the defined criteria and after short term finetuning the model, the accuracy is measured. Finally, the layer with highest accuracy is pruned. This process is repeated until total latency reduction satisfies the constraint. When NetAdapt was applied on 75 % MobileNetV1 [Howard et al., 2017] with 224 input size of network it, it outperforms Automated Deep Compression and Acceleration with Reinforcement Learning (ADC) approach [He, Han, 2018] with 19.9 million lesser MACs (Multiply Accumulate operations) and 4.3 ms lower latency on Google Pixel 1, Mobile CPU for the same accuracy of 69.1 % [Yang et al., 2018].

Automated pruning using reinforcement learning

He et al. came up with Automated Deep Compression (ADC) that aimed to achieve pruning and quantization using reinforcement learning, automating the compression process [He, Han, 2018]. This approach with later revised with AutoML for Model Compression (AMC) focusing only on pruning using reinforcement learning instead of relying on manually handcrafting rule-based approach for pruning. This approach reduces the overhead and effort involved in manually redesigning the network. A MobileNetV1 model pruned using AMC with an accuracy of 70.2 % takes 63 ms inference time as compared to a MobileNetV1 model pruned using NetAdapt with similar accuracy which takes around 80.35 ms inference time.

The comparison of ADC with NetAdapt and AMC with NetAdapt is shown in Fig. 9.

Hardware support for pruning

As the sparsity becomes crucial for parameters reduction and accelerated execution, NVIDIA Ampere GPU architectures stepped in to make sparsity adoption practical [Mishra et al., 2021]. NVIDIA introduced sparsity support in its matrix-math units, Tensor Cores. Tensor Cores are

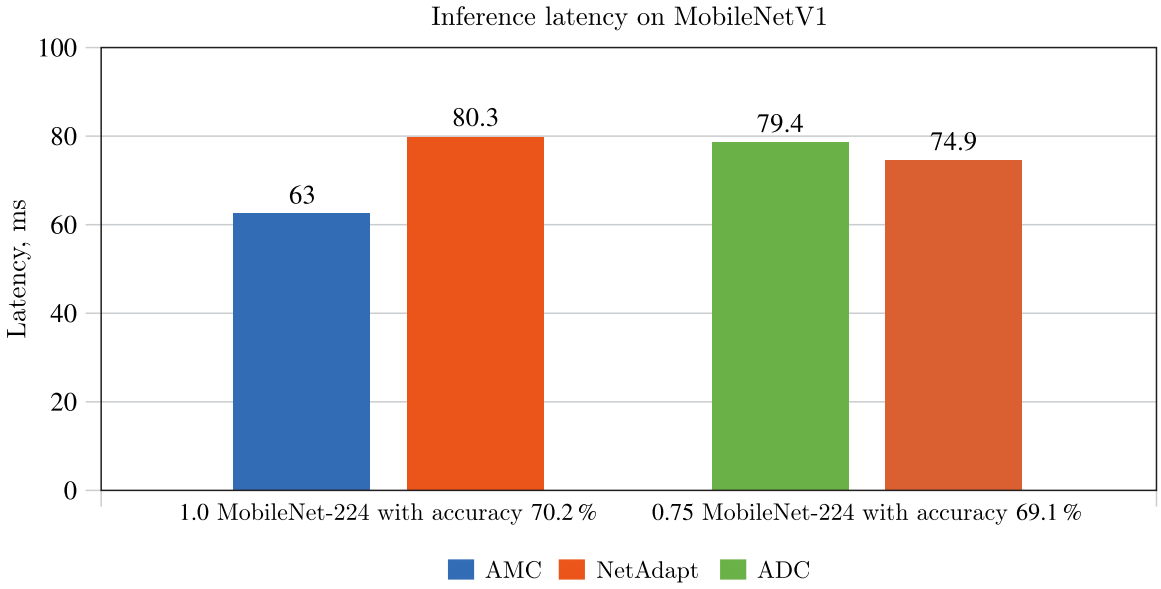

Figure 9. Comparison of pruning using ADC with NetAdapt and AMC with NetAdapt

specialized processing units within NVIDIA's GPUs, designed to accelerate the operations of deep learning models, especially matrix-multiplications. With the concept of $M : N$ sparsity, all the nonzero elements of matrix (M) out of total number of elements (N) are pushed to the left saving storage and computation and only the nonzero elements are used for matrix multiplication.

Qualcomm AI Research team compared Quantization with Pruning [Kuzmin et al., 2024] under some conditions. For pruning, magnitude pruning with fine-tuning was taken into consideration and for quantization, symmetric uniform quantization was considered. The results suggested that quantization outperformed the pruning technique in most cases.

Neural architecture search

A neural network can be designed in a certain way to achieve a particular objective. A very simple example can be a smaller size of model to be deployed on a lite device. But designing neural networks might require excessive involvement of human expertise. To counter this, Neural Architecture Search (NAS) comes to help. It is a way to design an architecture using AI approaches.

In 2019, Elsken et al. presented an overview of work done regarding the automated neural architecture search [Elsken, Metzen, Hutter, 2019]. They divided the methods NAS according to three dimensions: search space, search strategy and performance estimation strategy. From a search space an architecture is selected through a particular search strategy and then passed to performance estimation strategy. The performance estimate of that architecture is returned to search strategy and the process keeps repeating until an architecture is finalized.

Tiny NAS

Lin et al. proposed a two stage neural architecture search method named TinyNAS for better implementation of NAS on tiny devices [Lin et al., 2020]. It works by first optimizing the search space to cater the resource constraints and then with in that optimized search space, it performs neural architecture search. Using a simple heuristic that if a model has large number of FLOPs, it is more likely to have a higher accuracy and the models with higher FLOPs that can fit into the memory are better suited for the optimized search space.

Reinforcement learning based NAS

Zoph and Le used a Recurrent Network controller to generate a description of an architecture, and then a network with this architecture is trained [Zoph, Le, 2016]. The accuracy of the network is then used to update and improve the RNN controller using reinforcement learning.

Chen et al. developed two methods, namely, Net2WiderNet and Net2DeeperNet, in which a neural network learns from previous version of trained neural network with the help of knowledge transfer without the need of retraining the model again from scratch [Chen, Goodfellow, Shlens, 2016]. It also allows moving to a larger neural network and makes the NAS process easier and smoother.

Gradients based NAS

DARTS [Liu et al., 2018] is another technique for finding efficient architectures. It is based on a gradient-based method and treats the architecture search as a continuous problem, which also implies that the search space is considered a continuous range. In the search space, the architecture is represented as a directed acyclic graph and each node represents a feature while each edge represents an operation like Pooling, Convolution etc. Instead of using specific operation for an edge, DART uses combination of all operations and assigns weight to certain parameters. The key point about DARTS is that it handles both the convolutional as well as RNN architectures.

HyperNet based NAS

The SMASH technique [Brock et al., 2017] uses an assistant model — HyperNet, to generate weights for the neural network based on the design. Although the generated weights are not as good as the original weights of the neural networks, they help rank the architecture saving the training time of the neural network.

Zero-shot NAS

Measures like Gaussian complexity (Φ-scores) helps determine the expressitivity of a network i. e. how well a network can learn and represent different patterns and features. If networks are too deep and lack proper batch normalization layers, there are issues like numerical overflow. To address such challenges another measure called Zen-score was introduced, based on which a zero-shot method based on Zen-scores, known as Zen-NAS [Lin et al., 2021], was developed. It uses evolutionary search to generate neural architectures which only requires a few forward inferences, and does not require training of networks to find best architectures. This approach achieves top-1 accuracy on ImageNet-1k within half a GPU day outperforming approaches like DARTS, ProxylessNAS and NASNet-A which require around 4, 8.3 and 1800 GPU days respectively [Lin et al., 2021].

Hardware-aware NAS

While working with transformers it was observed that efficiency measurements based on FLOPs do not always reflect actual performance. The requirements for a transformer design change based on which hardware is being used. Using Hardware-Aware Transformers (HAT) [Wang et al., 2020], a SuperTransformer is trained that has multiple subnetworks and allows a search from different latencies to find best suited architecture for a hardware.

Since measuring inference latency on multiple devices is slow and expensive, an approach to predict the latency was introduced in ProxylessNAS [Cai, Zhu, Han, 2018]. A predictive model is trained to estimate the inference latency on various models, and after choosing the best model, the latency was measured to confirm the results. ProxylessNAS has around 200 GPU hours search cost on ImageNet with around 80 ms mobile latency while NASNet-A has around 48 000 GPU hours search cost with a latency of 183 ms and MobileNetV1 which is manually designed network has a latency of 113 ms [Cai, Zhu, Han, 2018].

In 2021, Neural Accelerator Architecture Search (NAAS) [Lin, Yang, Han, 2021] technique was proposed. This search technique iteratively searches and optimizes neural network structure and the hardware accelerator design it runs on. Along with that, it also searches for a compiler mapping that

maps the neural network architecture on a particular hardware accelerator. The hardware accelerator design mainly includes parameters like the size of the compute array, buffer sizes (for inputs, weights, and outputs), and the interconnections between processing elements.

Knowledge distillation

Knowledge distillation is a technique used in machine learning to transfer knowledge from a larger, more complex model (the "teacher") to a smaller, more efficient model (the "student"). Training through a teacher model in knowledge distillation helps the student model learn from the teacher's complex patterns and soft targets (probability distributions over classes predicted by teacher network) instead of hard targets (definitive class labels), improving its performance and efficiency.

Standard distillation

Hinton et al. uses an interesting analogy of larvae and adult periods of insect to explain that each period of lifecycle is designed to match certain needs [Hinton, Vinyals, Dean, 2015]. Similarly, we can divide machine learning tasks mainly into training and inference periods where training requires more focus on learning from the patterns and computational efficiency is usually not a concern, while for inference efficiency is crucial. It also introduces the concept of *distillation* where a big model is trained which is good at learning the patterns but the training and inference process is slow. When the soft labels from this big network are used to train a smaller network, the smaller network is observed to train faster and better, and it is also capable of performing inference faster because of less computations required.

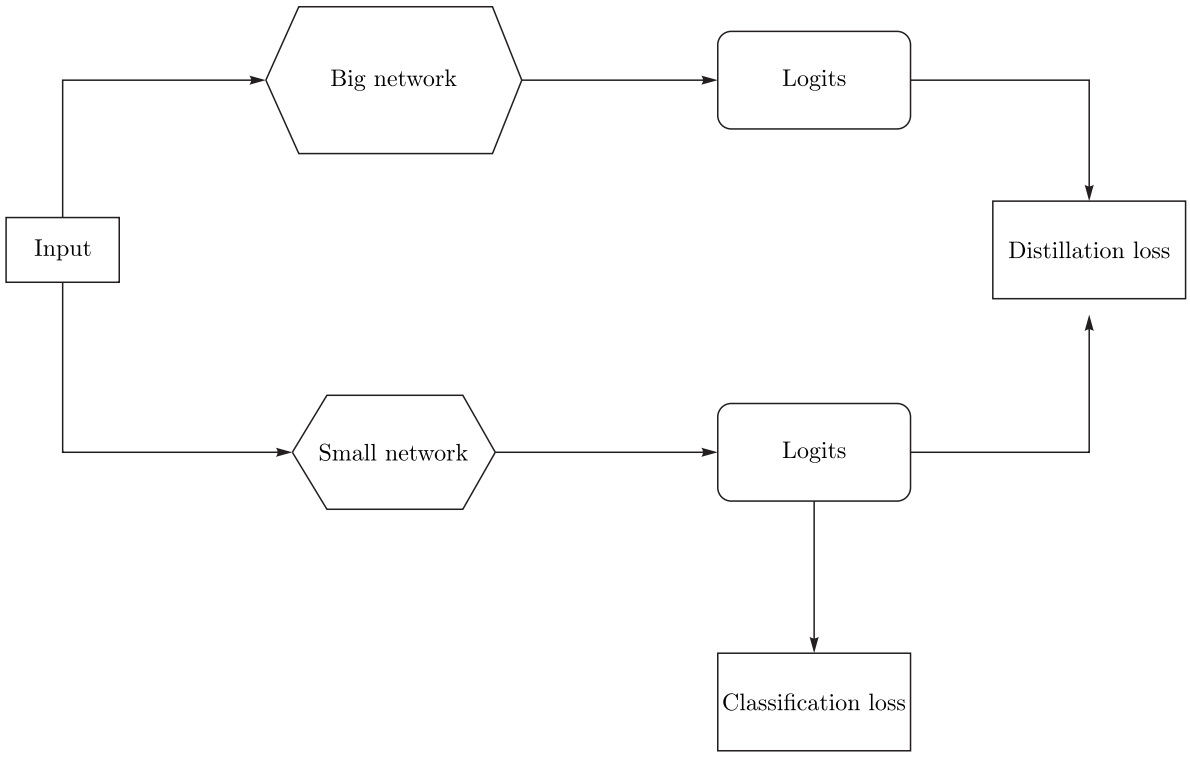

Figure 10. An illustration of Knowledge Distillation where small network learns from big network

Advanced knowledge distillation

FitNets [Romero et al., 2015] are a method to train smaller student networks using the intermediate weights of teacher network. Usually, the teacher network is wider, so a fully connected

layer or 1-by-1 convolutional layer by a linear transformation makes it possible for both networks to be compared.

Neurons in a network responds to certain patterns in the data. And if the neurons are activated by certain patterns, those patterns are important for the task. Neuron Selectivity Transfer (NST) [Huang, Wang, 2017] uses a method Maximum Mean Discrepancy to measure and minimize the difference between the activation patterns or feature maps of student and teacher networks.

While Neuron Selectivity Transfer focuses on neuron response patterns, another method which is about Attention Transfer [Zagoruyko, Komodakis, 2016] focuses more on transferring the attention information in CNNs from the teacher network to the student model, by comparing the attention map of student network and ensuring that they match the attention map of teacher network. This method is especially helpful for training CNNs.

Similarly, instead of focusing on neuron response as proposed in Neuron Selectivity Transfer, another approach [Heo et al., 2018] is to transfer activation boundaries formed by hidden layers, by looking if the neurons are activated or not. In case the teacher and student networks have different number of neurons, a connector function is used to map them. And it eventually allows the method to work even when architectures are different. This method can be applied to CNNs as well by considering hidden neuron responses as spatial dimensions. The loss function is adjusted to sum over all spatial locations, and a 1×1 convolutional layer is used as the connector function.

Relational knowledge distillation

In another approach [Yim et al., 2017], the student network instead of learning the intermediate results as proposed in FitNets, learns the method of solution. The flow of solution procedure (FSP) Matrix captures the flow of information by representing the relationship between features extracted by layers on deep neural network. It calculates how the information from one layer influences another layer by computing the inner products across layers, the FSP matrix effectively captures how the features (or activations). High values in the FSP matrix indicate strong influence, which means that the features extracted in one layer are critical for the other layer, while low values represent weak relationship. This approach provides better generalization to student network than teaching the intermediate results.

In Relational Knowledge Distillation [Park et al., 2019], the relationship between the outputs are also taken into consideration i. e. instead of individual mapping of outputs between teacher and student models, this approach focuses on transferring the relational structure of the outputs. Relational Knowledge Distillation introduces two main loss functions: Distance-wise Loss and Angle-wise Loss. Distance-wise loss function calculates the Euclidean distance between the outputs of teacher and student models which helps student learn the relative distances between outputs. And Angle-wise loss function measures the angles between triples of outputs which helps transfer more complex information as they capture the relative orientation and structure of the data in high-dimensional space. Figure 11 shows the difference in outputs mapping in conventional knowledge distillation and relational knowledge distillation.

Conclusions

In conclusion, deploying neural networks on lite devices is a fast-growing field with many new developments. The discussed techniques help reduce the size and computations required for neural networks ensuring that models can run efficiently on devices with limited power and processing capabilities. The recent developments focus more on automation techniques for implementing these methods reducing the need for manual intervention and making the process more scalable. Further advancements in these areas can include optimization techniques that are more automated and adaptable to different hardware platforms and emerging hardware innovations, such as specialized AI chips. Moreover, the discussed techniques can be refined to maintain accuracy as models become smaller and faster.

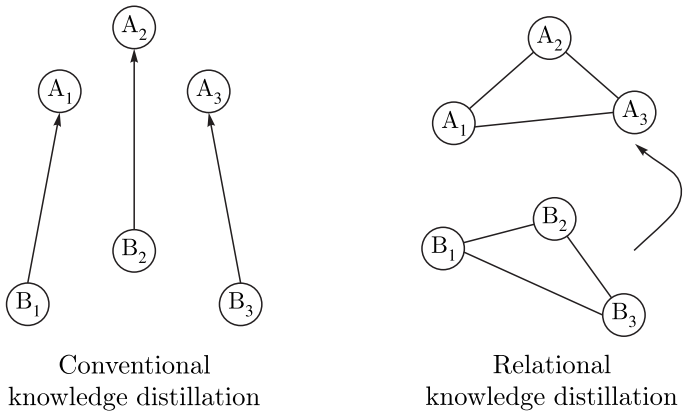

Figure 11. A representation of mapping between outputs in conventional knowledge distillation and outputs in relational knowledge distillation

References

Ahn H., Chen T., Alnaasan N., Shafi A., Abduljabbar M., Subramoni H., Panda D. K. Performance characterization of using quantization for DNN inference on edge devices: extended version // arXiv preprint. — 2023. — arXiv:2303.05016

Argüello Ron D., Freire P. J., Prilepov J. E. Performance evaluation of quantized neural networks on FPGAs // 2022.

Banner R., Nahshan Y., Hoffer E., Soudry D. ACIQ: analytical clipping for integer quantization of neural networks // CoRR. — 2018. — https://arxiv.org/abs/1810.05723v1

Bengio Y., Léonard N., Courville A. C. Estimating or propagating gradients through stochastic neurons for conditional computation // CoRR. — 2013. — http://arxiv.org/abs/1308.3432

Bondarenko Y., Nagel M., Blankevoort T. Quantizable transformers: removing outliers by helping attention heads do nothing // arXiv preprint. — 2023. — https://arxiv.org/abs/2306.12929

Brock A., Lim T., Ritchie J. M., Weston N. SMASH: one-shot model architecture search through hypernetworks // CoRR. — 2017. — http://arxiv.org/abs/1708.05344

Cai H., Zhu L., Han S. ProxylessNAS: direct neural architecture search on target task and hardware // CoRR. — 2018. — http://arxiv.org/abs/1812.00332

Capotondi A., Rusci M., Fariselli M., Benini L. CMix-NN: mixed low-precision CNN library for memory-constrained edge devices // IEEE Transactions on Circuits and Systems II: Express Briefs. — 2020. — Vol. 67, No. 5. — P. 871–875. — DOI: 10.1109/TCSII.2020.2983648

Chen T., Goodfellow I., Shlens J. Net2Net: accelerating learning via knowledge transfer // arXiv preprint. — 2016. — arXiv:1511.05641

Dai S., Venkatesan R., Ren H., Zimmer B., Dally W. J., Khailany B. VS-quant: per-vector scaled quantization for accurate low-precision neural network inference // CoRR. — 2021. — https://arxiv.org/abs/2102.04503

Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L. Imagenet: a large-scale hierarchical image database // 2009 IEEE conference on computer vision and pattern recognition. — 2009. — P. 248–255.

Drachman D. A. Do we have brain to spare? // Neurology. — 2005. — Vol. 64, No. 12. — P. 2004-5. — DOI: 10.1212/01.WNL.0000166914.38327.BB

Elsken T., Metzen J. H., Hutter F. Neural architecture search: A survey // arXiv preprint. — 2019. — arXiv:1808.05377

Famili A., Lao Y. Deep neural network quantization framework for effective defense against membership inference attacks // Sensors. — 2023. — Vol. 23, No. 18. — P. 7722. — DOI: 10.3390/s23187722

Garofalo A., Rusci M., Conti F., Rossi D., Benini L. PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors // Philosophical Transactions of the Royal Society A. — 2020. — DOI: 10.1098/rsta.2019.0155

Gholami A., Kim S., Dong Z., Yao Z., Mahoney M. W., Keutzer K. A survey of quantization methods for efficient neural network inference // arXiv preprint. — 2021. — arXiv:2103.13630

Han Q., Hu Y., Yu F., Yang H., Liu B., Hu P., Gong R., Wang Y., Wang R., Luan Z., Qian D. Extremely low-bit convolution optimization for quantized neural network on modern computer architectures // Proceedings of the 49th International Conference on Parallel Processing. — 2020. — P. 38. — DOI: 10.1145/3404397.3404407

Han S., Pool J., Tran J., Dally W.J. Learning both weights and connections for efficient neural networks // CoRR. — 2015. — http://arxiv.org/abs/1506.02626

Han S., Mao H., Dally W.J. Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding // arXiv preprint. — 2016. — https://arxiv.org/abs/1510.00149

Hawks B., Duarte J., Fraser N. J., Pappalardo A., Tran N., Umuroglu Y. Ps and Qs: quantization-aware pruning for efficient low latency neural network inference // Frontiers in Artificial Intelligence. — 2021.

He Y., Han S. ADC: automated deep compression and acceleration with reinforcement learning // CoRR. — 2018. — http://arxiv.org/abs/1802.03494

He Y., Zhang X., Sun J. Channel pruning for accelerating very deep neural networks // CoRR. — 2017. — http://arxiv.org/abs/1707.06168

Heo B., Lee M., Yun S., Choi J. Y. Knowledge transfer via distillation of activation boundaries formed by hidden neurons // CoRR. — 2018. — http://arxiv.org/abs/1811.03233

Hinton G., Vinyals O., Dean J. Distilling the knowledge in a neural network // arXiv preprint. — 2015. — https://arxiv.org/abs/1503.02531

Horowitz M. 1.1 Computing's energy problem (and what we can do about it) // 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). — 2014. — P. 10–14. — DOI: 10.1109/ISSCC.2014.6757323

Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M., Adam H. MobileNets: efficient convolutional neural networks for mobile vision applications // CoRR. — 2017. — http://arxiv.org/abs/1704.04861

Hu H., Peng R., Tai Y.-W., Tang C.-K. Network trimming: a data-driven neuron pruning approach towards efficient deep architectures // CoRR. — 2016. — http://arxiv.org/abs/1607.03250

Huang Z., Wang N. Like what you like: knowledge distill via neuron selectivity transfer // CoRR. — 2017. — http://arxiv.org/abs/1707.01219

Huttenlocher P. R. Morphometric study of human cerebral cortex development // Neuropsychologia. — 1990. — Vol. 28, No. 6. — P. 517–527.

Jacob B., Kligys S., Chen B., Zhu M., Tang M., Howard A. G., Adam H., Kalenichenko D. Quantization and training of neural networks for efficient integer-arithmetic-only inference // CoRR. — 2017. — http://arxiv.org/abs/1712.05877

Jain A., Bhattacharya S., Masuda M., Sharma V., Wang Y. Efficient execution of quantized deep learning models: a compiler approach // arXiv preprint. — 2020. — arXiv:2006.10226

Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper // CoRR. — 2018. — http://arxiv.org/abs/1806.08342

Krizhevsky A., Hinton G. Learning multiple layers of features from tiny images. — Toronto, ON, Canada, 2009.

Kuzmin A., Nagel M., van Baalen M., Behboodi A., Blankevoort T. Pruning vs quantization: which is better? // arXiv preprint. — 2024. — https://arxiv.org/abs/2307.02973

LeCun Y., Denker J., Solla S. Optimal brain damage // Advances in Neural Information Processing Systems / D. Touretzky (Ed.). — 1989. — Vol. 2.

Lee J., Yu M., Kwon Y., Kim T. Quantune: Post-training quantization of convolutional neural networks using extreme gradient boosting for fast deployment // Future Generation Computer Systems. — 2022. — Vol. 132. — P. 124–135. — DOI: 10.1016/j.future.2022.02.005

Li P., Yang J., Islam M. A., Ren S. Making AI less "thirsty": uncovering and addressing the secret water footprint of AI models // arXiv. — 2023. — arXiv:2304.03271

Li T., Ma Y., Endoh T. From algorithm to module: adaptive and energy-efficient quantization method for edge artificial intelligence in IoT society // IEEE Transactions on Industrial Informatics. — 2023. — Vol. 19, No. 8. — P. 8953–8964.

Liang T., Glossner J., Wang L., Shi S., Zhang X. Pruning and quantization for deep neural network acceleration: a survey // arXiv preprint. — 2021. — arXiv:2101.09671

Lin J., Chen W.-M., Lin Y., Cohn J., Gan C., Han S. MCUNet: tiny deep learning on IoT devices // CoRR. — 2020. — https://arxiv.org/abs/2007.10319

Lin J., Zhu L., Chen W. M., Wang W. C., Han S. Tiny machine learning: progress and futures [feature] // IEEE Circuits and Systems Magazine. — 2023. — Vol. 23, No. 3. — P. 8–34. — DOI: 10.1109/MCAS.2023.3302182

Lin M., Wang P., Sun Z., Chen H., Sun X., Qian Q., Li H., Jin R. Zen-NAS: a zero-shot NAS for high-performance deep image recognition // CoRR. — 2021. — https://arxiv.org/abs/2102.01063

Lin Y., Yang M., Han S. NAAS: neural accelerator architecture search // CoRR. — 2021. — https://arxiv.org/abs/2105.13258

Liu H., Simonyan K., Yang Y. DARTS: differentiable architecture search // CoRR. — 2018. — http://arxiv.org/abs/1806.09055

Liu Z., Li J., Shen Z., Huang G., Yan S., Zhang C. Learning efficient convolutional networks through network slimming // CoRR. — 2017. — URL: http://arxiv.org/abs/1708.06519

Mao H., Han S., Pool J., Li W., Liu X., Wang Y., Dally W. J. Exploring the granularity of sparsity in convolutional neural networks // 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). — 2017. — P. 1927–1934. — DOI: 10.1109/CVPRW.2017.241

Mishra A. K., Latorre J. A., Pool J., Stosic D., Stosic D., Venkatesh G., Yu C., Micikevicius P. Accelerating sparse deep neural networks // CoRR. — 2021. — https://arxiv.org/abs/2104.08378

Nagel M., van Baalen M., Blankevoort T., Welling M. Data-free quantization through weight equalization and bias correction // CoRR. — 2019. — http://arxiv.org/abs/1906.04721

Novac P.-E., Boukli Hacene G., Pegatoquet A., Miramond B., Gripon V. Quantization and deployment of deep neural networks on microcontrollers // Sensors. — 2021. — Vol. 21, No. 9. — DOI: 10.3390/s21092984

Park W., Kim D., Lu Y., Cho M. Relational knowledge distillation // CoRR. — 2019. — http://arxiv.org/abs/1904.05068

Rokh B., Azarpeyvand A., Khanteymoori A. A comprehensive survey on model quantization for deep neural networks in image classification // ACM Trans. Intell. Syst. Technol. — 2023. — Vol. 14, No. 6. — P. 97. — DOI: 10.1145/3623402

Romero A., Ballas N., Ebrahimi Kahou S., Chassang A., Gatta C., Bengio Y. FitNets: hints for thin deep nets // arXiv preprint. — 2015. — https://arxiv.org/abs/1412.6550

Rouhani B., Zhao R., Elango V., Shafipour R., Hall M., Mesmakhosroshahi M., More A., Melnick L., Golub M., Varatkar G., Shao L., Kolhe G., Melts D., Klar J., L'Heureux R., Perry M., Burger D., Chung E., Deng Z., Naghshineh S., Park J., Naumov M. With shared microexponents, a little shifting goes a long way // arXiv preprint. — 2023. — https://arxiv.org/abs/2302.08007

Rusci M., Capotondi A., Benini L. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers // CoRR. — 2019. — http://arxiv.org/abs/1905.13082

Sakr C., Dai S., Venkatesan R., Zimmer B., Dally W.J., Khailany B. Optimal clipping and magnitude-aware differentiation for improved quantization-aware training // arXiv preprint. — 2022. — https://arxiv.org/abs/2206.06501

Srinivas S., Kuzmin A., Nagel M., van Baalen M., Skliar A., Blankevoort T. Cyclical pruning for sparse neural networks // CoRR. — 2022. — https://arxiv.org/abs/2202.01290

Vaswani A. Attention is all you need // Advances in Neural Information Processing Systems. — 2017.

Verma G., Gupta Y., Malik A.M., Chapman B. Performance evaluation of deep learning compilers for edge inference // 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). — 2021. — P. 858–865.

Walsh C.A. Peter Huttenlocher (1931–2013) // Nature. — 2013. — Vol. 502, No. 7470. — P. 172. — DOI: 10.1038/502172a

Wang H., Wu Z., Liu Z., Cai H., Zhu L., Gan C., Han S. HAT: hardware-aware transformers for efficient natural language processing // CoRR. — 2020. — arXiv:2005.14187

Wang P., Chen W., He X., Chen Q., Liu Q., Cheng J. Optimization-based post-training quantization with bit-split and stitching // IEEE Transactions on Pattern Analysis & Machine Intelligence. — 2023. — Vol. 45, No. 02. — P. 2119–2135. — DOI: 10.1109/TPAMI.2022.3159369

Wen W., Wu C., Wang Y., Chen Y., Li H. Learning structured sparsity in deep neural networks // CoRR. — 2016. — http://arxiv.org/abs/1608.03665

Wu H., Judd P., Zhang X., Isaev M., Micikevicius P. Integer quantization for deep learning inference: principles and empirical evaluation // arXiv. — 2020. — https://arxiv.org/abs/2004.09602

Yang T.-J., Howard A. G., Chen B., Zhang X., Go A., Sze V., Adam H. NetAdapt: platform-aware neural network adaptation for mobile applications // CoRR. — 2018. — http://arxiv.org/abs/1804.03230

Yim J., Joo D., Bae J., Kim J. A gift from knowledge distillation: fast optimization, network minimization and transfer learning // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2017. — P. 7130–7138. — DOI: 10.1109/CVPR.2017.754

Zagoruyko S., Komodakis N. Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer // CoRR. — 2016. — http://arxiv.org/abs/1612.03928

Zhu C., Han S., Mao H., Dally W.J. Trained ternary quantization // arXiv. — 2017. — https://arxiv.org/abs/1612.01064

Zoph B., Le Q.V. Neural architecture search with reinforcement learning // CoRR. — 2016. — http://arxiv.org/abs/1611.01578