# Neural Network Optimization For Edge Device

Oussama Fezzai
*University Paris-Saclay, Évry Val d'Essonne*
*UFR Sciences Technologies*
*Électronique, Énergie Électrique, Automatique*

***Abstract*— While Deep Neural Networks (DNNs) have demonstrated significant benefits across diverse domains, their applicability to edge devices faces limitations, necessitating a shift from cloud-based solutions. Cloud computing offers high accuracy but introduces challenges such as potential long latency and privacy concerns associated with data transmission. This is particularly problematic in remote areas where the reliability of connections between edge devices and remote servers is compromised.**

**To address these challenges, this paper advocates for the optimization of DNNs on edge devices through the application of four keys techniques: Quantization, Pruning,Knowledge distillation and Neural Network Architecture Search. Quantization involves converting floating-point model parameters to integers, enabling fixed-point arithmetic and reducing computational demands. Pruning identifies and removes redundant neurons, reducing computation and space complexity with minimal impact on accuracy. Knowledge distillation by transfer knowledge from a larger or more complex model (teacher model) to a smaller or simpler model (student model) . Neural Network Architecture Search focuses on designing novel DNN architectures tailored for mobile and IoT devices, enhancing computation and storage efficiency on edge devices.**

**In this paper, each optimization method is comprehensively explained, detailing their respective advantages and disadvantages. The research also provides insights into recent developments in this evolving field, shedding light on the ongoing efforts to overcome challenges and improve the deployment of DNNs on edge devices.**

## I. INTRODUCTION

In recent years, the ubiquity of Deep Neural Networks (DNNs) has revolutionized numerous domains, showcasing their prowess in achieving unprecedented levels of accuracy and performance. However, the deployment of these powerful neural networks faces significant challenges when applied to edge devices, prompting a reevaluation of traditional cloud-based paradigms. While cloud computing offers the advantage of leveraging high-performance DNN models on remote servers, the associated drawbacks, such as potential long latency and privacy concerns, become pronounced when applied to edge computing.

The inherent limitations of deploying DNNs on edge devices stem from the intricate balance required between computational efficiency and the demand for high accuracy. This is especially pertinent in scenarios where reliable connections between edge devices and remote servers are compromised, as is often the case in rural areas, where factories and warehouses are commonly situated.

This paper delves into the intricacies of optimizing neural networks for edge devices, focusing on four fundamental techniques: Quantization, Pruning, Knowledge distillation and Neural Network Architecture Search. These techniques aim to address the challenges posed by the limitations of edge computing, offering a comprehensive solution to enhance the efficiency of DNNs in resource-constrained environments. Each method is explored in detail, emphasizing their advantages, disadvantages, and recent advancements in the dynamic field of neural network optimization for edge devices. Through this exploration, the paper seeks to contribute to the ongoing discourse surrounding the evolution of DNN deployment, offering insights into the promising strides made and the future potential of neural networks in edge computing scenarios. This paper is organized as follows.

The first part delves into an exploration of various techniques for optimizing neural networks. Each

technique is comprehensively explained, elucidating its workings and highlighting the essential elements involved.
The second part addresses challenges encountered in the optimization process and outlines potential avenues for future research and development.

## II. Various Techniques

Some of the various techniques being practiced worldwide for model compression and acceleration are discussed below.

### A. Qunatization

To address the intricate computational and spatial challenges inherent in neural network models, a prevalent strategy is the application of quantization techniques for model compression. This method entails representing each weight and activation in the neural network using a reduced number of bits. By substituting floating-point numbers with fixed-point counterparts, arithmetic operations gain notable efficiency in terms of resource utilization and power consumption when compared to their floating-point equivalents. This not only leads to a considerable reduction in hardware complexity but also curtails the number of off-chip memory accesses, resulting in significant cost savings in communication. A crucial component of this strategy is parameter quantization, involving the conversion of weights and activation values in a network model from high precision to low precision[1]. The quantization approach use algorithm 1

for find S and Z we use these two equations:

$$S = \frac{R_{\max} - R_{\min}}{Q_{\max} - Q_{\min}} \tag{1}$$

$$Z = Q_{\max} - \frac{R_{\max}}{S} \tag{2}$$

where $R$ denotes a real floating-point number, $Q$ denotes the quantization fixed-point value, $Z$ denotes the quantization fixed-point value corresponding to the zero floating-point value, and $S$ is the scale factor of quantization [2].

There are a lot of techniques of quantization :

**Algorithm 1:** Parametre Quantization

**Input :** Input data (weights and activation function)
**Output:** Quantized model
**Step 1:** Count the corresponding *min_value* and *max_value* in the input data (weights and activation function);
**Step 2:** Choose the appropriate quantization type, symmetric (*int8*) or asymmetric (*uint8*);
**Step 3:** Calculate the quantization parameters $Z$/Zero point and $S$/Scale according to the quantization type, *min_value*, and *max_value*;
**Step 4:** Quantize the model based on the calibration data, converted from *FP32* to *int8*;
**Step 5:** Verify the performance of the quantized model, and if the result is not satisfactory, try a different way to calculate $S$ and $Z$, then re-execute the above operations;

*1) Binarized Quantization:* Quantization approach which quantizes each weight and activation of a neural network to binary value,It reduces memory and memory access by 32X. Also, the time complexity of the convolution operation is reduced by 60%[3].

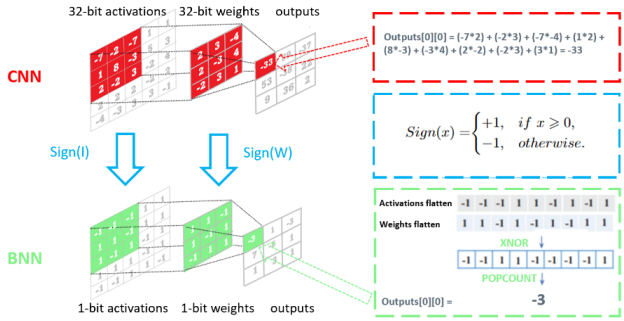


FIG 1:Binarized convolution neural network [1].

*2) Multi-bit Quantization:* Multi-bit Quantization is a method for improving Binarized quantization by $R = S(Q - Z)$.

The equation you've provided is a simple form of quantization with scaling. Let's break down the components:

$R$ : This represents the original continuous or

floating point value you want to quantize.

$S$ : This is the scaling constant. It's used to scale the difference between the original value (R) and the quantized integer value (Q).

$Q$ : This is the quantized integer value for r. It's the output of the quantization process.

$Z$ : This is the quantized integer value for zero (0).

The equation says that to obtain the quantized integer value Q for the original value R, you subtract the quantized integer value for zero (Z) from the original value (R) and then scale the result by the scaling constant (S).

Mathematically, this equation is used to map a continuous or floating-point value (R) to a quantized integer value (Q). It's often used in quantization processes in digital signal processing and data compression. The goal of quantization is to represent the continuous value (R) with a quantized version (Q) that uses a limited number of bits, which is more memory-efficient.

The choice of the scaling constant (S) and the quantization for zero (Z) is essential in determining the precision and range of the quantization. Different applications may use different scaling constants and zero values to achieve the desired quantization properties.

*3) Comparison:* Binarized quantization significantly reduces the computation and space complexity of a DNN model.The multi-bit quantization uses more bits to represent weights and activation compared to the binarized quantization, but the accuracy is much higher than the binarized quantization, especially on complicated datasets such as the ImageNet dataset. Besides the image classification task, the multi-bit quantization approach has been extended to other tasks such as object detection, face detection, and face attributes.

After quantization, the parameters of the model usually need to be adjusted. The process of obtaining a model by retraining is called quantization-aware training (QAT). Similarly, the process of obtaining a model without retraining is called post-training quantization (PTQ).

### QAT: Quantization-Aware Training

Quantization-Aware Training is a strategy that involves training a neural network while taking into account the quantization effects that will be applied during inference. The goal is to make the model more robust to the quantization process, ensuring that its performance is not significantly degraded when deployed on hardware with reduced precision, such as low-bit fixed-point representations[4].

### PTQ: Post-Training Quantization

In this case, we let the model undergo training, and after that, we select a small set of training data, referred to as calibration data, to find the proper range and scaling factor for quantization. After this calibration step, we deploy the model to an edge device[4].

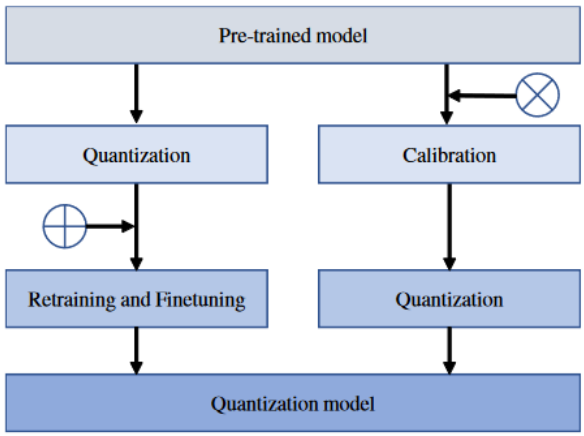


FIG 2:the schema of process QAT is in the Right and PTQ is in the Left[2].

## B. Pruning

Pruning is a technique intended to automatically find redundant and remove neurons and connections for model compression and acceleration [5]. All the Pruning techniques use the algorithm 2

*1) Pruning Structured:* Pruning strctured is type of pruning NN by removing channel and the corresponding channels of this channel this implies we remove features theirs are not importants ,this approach helps control the level of sparsity introduced to the model and may contribute to preserving certain architectural properties during the pruning process.

**Algorithm 2:** Pruning and Fine-Tuning

**Input :** $N$, the number of iterations of pruning, and $X$, the dataset on which to train and fine-tune

**Output:** $M$, the pruned model, and $W$, the fine-tuned model

$W \leftarrow$ initialize();
$W \leftarrow$ trainToConvergence$(f(X, W))$;
$M \leftarrow |W|$;
**for** $i \leftarrow 1$ **to** $N$ **do**
  $M \leftarrow$ prune$(M,$ score$(W))$;
  $W \leftarrow$ fineTune$(f(X, MW))$;
**return** $M$, $W$;

*2) Pruning Un-Structured:* Pruning unstructured mean removing individual weights or parameters from a neural network without any specific pattern or structure. The focus is on identifying and zeroing out weights that are considered unimportant for the model's performance.the unstructured pruning use no regular sparsity

Pruning methods vary primarily in their choices regarding sparsity structure, scoring, scheduling, and fine-tuning.

**Sparsity Structure:** This refers to which parts of the neural network are pruned. Different methods might focus on pruning entire neurons, individual weights, or even entire layers. The sparsity structure chosen can impact the efficiency of the pruned model and its performance.

**Scoring:** Scoring is the criteria used to determine which weights or connections are pruned. Common scoring methods include magnitude-based methods (pruning small weights), sensitivity-based methods (measuring the impact of each weight on the output), and more complex criteria that take into account factors like second-order information or the importance of weights during training.

**Scheduling:** Scheduling relates to when and how pruning is applied during the training process. Pruning can be done iteratively, with multiple rounds of training and pruning, or it can be done in a one-shot manner. Scheduling decisions can affect the balance between model size reduction and maintaining or improving performance.

**Fine-tuning:** After pruning, the pruned model is often fine-tuned to recover some of the performance lost during the pruning process. Fine-tuning involves retraining the pruned model on the original task, usually with a lower learning rate to allow the remaining weights to adjust to compensate for the pruned connections.

Different combinations of these choices can lead to various pruning methods. Some popular pruning methods include:

- **Weight Pruning:** Removing individual weights based on their magnitude.
- **Filter or Neuron Pruning:** Removing entire neurons or filters.
- **Layer Pruning:** Removing entire layers of the neural network.
- **Iterative Pruning:** Repeating the pruning and fine-tuning process multiple times.
- **Global Pruning:** Pruning a fixed percentage of weights globally across the entire network.

Each method has its advantages and disadvantages, and the choice of the pruning technique may depend on factors such as the specific neural network architecture, the dataset, and the computational resources available.

*3) Evaluating Pruning:* Pruning can accomplish many different goals, including reducing the storage footprint of the neural network, the computational cost of inference, the energy requirements of inference, etc. Each of these goals favors different design choices and requires different evaluation metrics To quantify efficiency, most papers report at least one of two metrics. The first is the number of multiply-adds (often referred to as FLOPs) required to perform inference with the pruned network. The second is the fraction of parameters pruned. To measure quality, nearly all papers report changes in Top-1 or Top-5 image classification accuracy. Pruned models sometimes outperform the original architecture, but rarely outperform a better architecture.

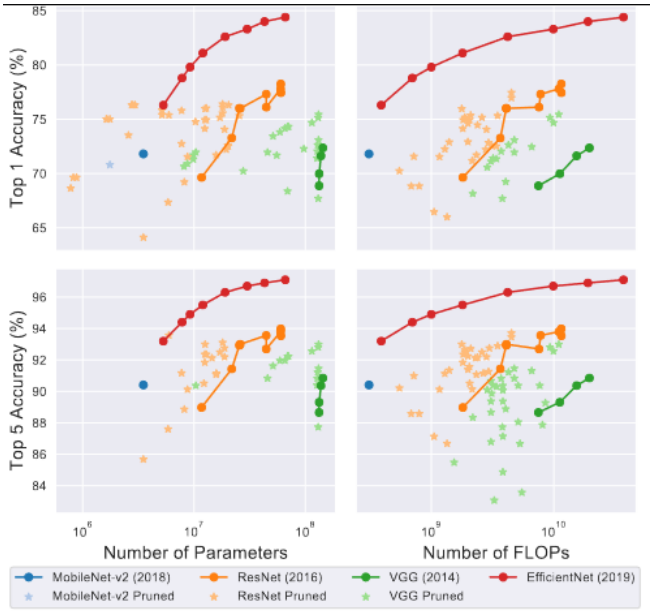


FIG 3: Comparison between pricipale model and purned model in same state of the art models [5].

### C. knowledge distillation

Distilling knowledge in a neural network, often referred to as knowledge distillation, is a technique used to transfer knowledge from a larger or more complex model (teacher model) to a smaller or simpler model (student model). This process helps improve the performance of the smaller model by leveraging the insights and information captured by the larger model [6].

The reasons for employing knowledge distillation are:

- Model compression
- Transfer learning
- Regularization

#### Overview of Knowledge Distillation

1) **Teacher Model:** Start with a well-trained teacher model, which can be a deep neural network, such as a large convolutional neural network (CNN) or a complex transformer model. This model serves as the source of knowledge[7].
2) **Student Model:** Create a smaller or simpler student model, often with fewer parameters, layers, or neurons. This model will be the one you intend to deploy for inference[7].
3) **Soft Targets:** Train the teacher model to produce "soft targets" in addition to the regular class predictions. These soft targets can be probability distributions over the classes rather than one-hot encoded labels. Soft targets contain more information about the relationships between classes and are used to transfer knowledge to the student model[7].
4) **Student Training:** Train the student model using the same dataset and objective as the teacher model but with a modified loss function. The loss function combines the standard cross-entropy loss for the hard labels (ground truth) and a term that measures the Kullback-Leibler (KL) divergence between the soft targets produced by the teacher model and the student's predictions[7].
5) **Temperature Scaling (Optional):** Introduce a temperature parameter in the softmax layer of both the teacher and student models. Temperature scaling can be used to control the softness of the probability distributions. A higher temperature makes the distributions softer, which can help the student model learn more effectively from the teacher[7]. The modified softmax function with temperature scaling is given by:

$$P_i = \frac{\exp(\frac{z_i}{T})}{\sum_{j=1}^{N} \exp(\frac{z_j}{T})}$$

Here:

- $P_i$ is the softened probability of class $i$,
- $z_i$ is the logit (raw score) of class $i$ from the model,
- $T$ is the temperature parameter,
- $\sum_{j=1}^{N}$ represents the sum over all classes.

6) **Fine-Tuning (Optional):** After the initial distillation, further fine-tune the student model on the target task to enhance its performance.

In knowledge distillation, there are three types [2]:

#### 1. Response-Based Knowledge

The main idea is to directly mimic the final prediction of the teacher network. The response-based knowledge is called a soft target, which is the probability of different classes of inputs that can be estimated by the Softmax function with a higher temperature. The softmax probabilities are more evenly distributed among the elements,

making the distribution ”softer.” With a lower temperature, the probabilities become more focused on the element with the highest logit, resulting in a ”sharper” distribution.

### 2. Feature-Based Knowledge

In this case, the teacher model and student model are designed to compare feature maps between them by a loss function.

### 3. Relation-Based Knowledge

The relation-based knowledge method further explores the relationship between different layers. Knowledge distillation using relation-based knowledge extends the traditional knowledge distillation approach by incorporating additional information about relationships, dependencies, or structures present in the data. This can lead to more effective model compression and improved generalization in certain scenarios.

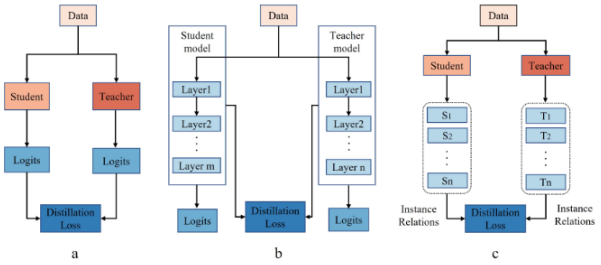


FIG 4:Threre typres of knowledge distillationt. (a) Response-Based Knowledge. (b) Feature-Based Knowledge. (c) Relation-Based Knowledge[6].

## D. Neural Architecture Search

The most basic definition of NAS is as follows. Given a search space $A$, a dataset $D$, a training pipeline $P$, and a time or computation budget $t$, the goal is to find an architecture $a \in A$ within budget $t$ which has the highest possible validation accuracy when trained using dataset $D$ and training pipeline $P$ [8].

We categorize methods for NAS according to three dimensions: search space, search strategy, and performance estimation strategy:

### 1. Search Space

The search space defines which architectures can be represented in principle. Incorporating prior knowledge about typical properties of architectures well-suited for a task can reduce the size of the search space and simplify the search. However, this also introduces a human bias, which may prevent finding novel architectural building blocks that go beyond the current human knowledge [8].

### 2. Search Strategy

The search strategy details how to explore the search space (which is often exponentially large or even unbounded). It encompasses the classical exploration-exploitation trade-off since, on the one hand, it is desirable to find well-performing architectures quickly, while on the other hand, premature convergence to a region of suboptimal architectures should be avoided [8].

Many different search strategies can be used to explore the space of neural architectures, including random search, Bayesian optimization, evolutionary methods, reinforcement learning (RL), and gradient-based methods [8].

### 3. Performance Estimation Strategy

The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data. Performance Estimation refers to the process of estimating this performance: by using lower fidelity estimate, supernet/weight sharing, weight inheritance, learning curve extrapolation.
the simplest option is to perform a standard training and validation of the architecture on data, but this is unfortunately computationally expensive and limits the number of architectures that can be explored. Much recent research therefore focuses on developing methods that reduce the cost of these performance estimations [8].

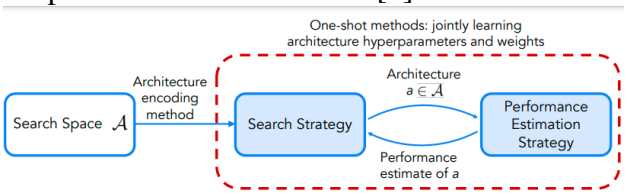


FIG 5: Process of neural architecture search [8].

Zoph et al In [9], they employ reinforcement learning for neural network search to generate child models, such as CNNs or RNNs, where each layer generates properties for each filter or other components. This approach not only yields models that outperform state-of-the-art models but also

provides superior performance compared to the best-performing models.

NAS is computationally expensive and time consuming training days. Meanwhile, using less resources tends to produce less compelling results. We observe that the computational bottleneck of NAS is the training of each child model to convergence, only to measure its accuracy whilst throwing away all the trained weights

In [10] to improve the efficiency of NAS by forcing all child models to share weights to eschew training each child model from scratch to convergence. The idea has apparent complications, as different child models might utilize their weights differently, but was encouraged by previous work on transfer learning and multitask learning, which established that parameters learned for a particular model on a particular task can be used for other models on other tasks

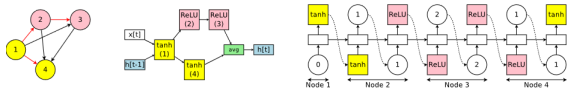


FIG 6: Efficient Neural Network Search via Parameter sharing [10].

### *E. Challenges and future works*

In model-pruning algorithms, most existing approaches focus on removing redundant connections or neurons from the network. However, this low-level pruning introduces unstructured risks. It is essential to propose more effective methods for evaluating the impact of pruned elements on model performance.

Parameter quantization is effective in reducing model size, but the quantization operation increases operational complexity. Special processing is required during quantization to mitigate severe accuracy loss. An appropriate quantization strategy aims to minimize complexity while preserving accuracy. Mixed accuracy quantization strategies contribute to reducing parameter size reasonably.

Knowledge distillation involves guiding the training of student networks by teacher networks. Training difficulty varies for different student network architectures, necessitating a richer theoretical foundation and experience in network engineering for designing student network architectures.

The neural network search algorithm is crucial for automatically designing networks by searching for the optimal architecture, but it remains time-consuming due to the training of each child model and its size is will be bottleneck for NAS.

Future work in neural network optimization for edge devices involves designing efficient architectures, exploring hardware-aware techniques, and advancing quantization and pruning methods. Researchers are also focusing on federated learning, dynamic inference, and energy-efficient, real-time inference. Transfer learning for edges, robustness, security, and the development of specialized AutoML tools are critical areas of exploration for optimizing neural networks in resource-constrained edge environments.

## III. Conclusion

In conclusion, the article navigates the landscape of optimizing neural networks for edge devices through techniques such as model pruning, parameter quantization, neural architecture search, and knowledge distillation. It highlights the need to assess the impact of model pruning and find a delicate balance in quantization complexity for accuracy preservation. The importance of automated neural architecture search and knowledge distillation in guiding student network training is emphasized. The article acknowledges challenges, particularly manual hyperparameter reliance, advocating for a standardized approach and paving the way for future research directions. Overall, it offers a comprehensive exploration of optimization strategies for edge device neural networks, emphasizing the crucial balance between efficiency and performance, while proposing avenues for ongoing research in this dynamic field.

## References


[1] Shiya Liu, Dong Sam Ha, Fangyang Shen, and Yang Yi. Efficient neural networks for edge devices. *Computers & Electrical Engineering*, 92:107121, 2021.

[2] Zhuo Li, Hengyi Li, and Lin Meng. Model compression for deep neural networks: A survey. *Computers*, 12(3):60, 2023.

[3] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[4] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[5] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

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

[7] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[8] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[9] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[10] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.