Proceedings of CONF-FMCE 2026 Symposium: Smart City and Infrastructure Engineering
DOI: 10.54254/2755-2721/2026.CH37554

Lightweight Neural Network Technology for Edge Computing

Ang Su

Henan Experimental High School, Zhengzhou, China
SUANGANDREW@outlook.com

Abstract. With the increase in computational requirements for deep learning, it has become difficult for these models to be applied to the development of edge devices in the Internet of Things; thus, lightweight networking technology has been one of the main research directions recently. This paper will present an organised list of light neural network technologies and introduce the three main model compression techniques: pruning, quantization and knowledge distillation. According to the actual engineering conditions, this paper will discuss the three main problems preventing the use of the model, analyze in detail the inherent deficiencies of the current technology: Trade-off between model accuracy and compression ratio, difficulty of edge hardware, limitation between theoretical and actual engineering application. Finally, it puts forward some specific suggestions to find practical ways to improve future lightweight networking technology for edge computing. The existing compression workflow must support more CPU architectures, also the collaborative software-hardware co-design research should continue, and simple edge evaluation should be introduced. This paper provides references for lightweight edge research.

Keywords: Lightweight neural networks, Edge computing, Embedded systems, Model compression

1. Introduction

1.1. Background

Deep learning, which has many layers and is inspired by the human brain, started to appear in 2012 and has been developing steadily ever since, showing excellent results in many industrial applications and other areas. At the bottom of deep learning is the conversion of raw data into abstract forms. According to research, an increase in the depth of a neural network will enhance its performance [1]; thus, in the beginning of deep learning, the focus was on improving the accuracy of models by continuously increasing both the depth and the number of parameters of the network. For example, classic models such as VGGNet and ResNet have many parameters and achieve high accuracy on ImageNet, but they also have high memory requirements and high computational complexity. Although large models of this scale can run without problems on servers, when deployed on resource-limited devices such as smartphones, cameras, and intelligent cars in mobile Internet and Internet of Things (IoT) environments, the strict constraints on storage space and

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.CH37554

processing power are severe bottlenecks, and running these models directly is not feasible. To address the problem of not being able to deploy traditional networks directly at the edge, lightweight networking technology has started to be introduced in edge computing. With the help of the edge, local processing can be performed to obtain a response time of milliseconds, reduce the amount of data that needs to be sent back to the central network, and prevent data leakage. Lightweight networking technology at the edge can reduce the number of model parameters and computation, and at the same time, minimize the loss of accuracy in the model; thus, it can operate efficiently and securely on resource-constrained devices [2- 4].

1.2. Research results

In recent years, the optimisation methods for neural networks at the edge have been divided into three categories: model-level optimisation, hardware-software co-optimisation and system-level optimization [3]. Among them, more attention has been paid to optimisation at the model level because it directly affects the network structure; its two main directions are manually designed native lightweight network structures and model compression methods [5]. YOLO is a typical series of algorithms, and as it has developed over time, both the backbone network and the neck network have been continuously improved. At the same time, model compression techniques such as pruning, quantization and knowledge distillation have also been applied to further optimise the YOLO model and reduce the number of parameters and computation. It will greatly reduce the memory and computation demands of inference, so now it is possible to deploy YOLO on edge devices with limited computing power; a lightweight YOLO model on an embedded edge computer can achieve good real-time performance and high security. YOLO models can also be used to process the videos from surveillance cameras in real time for traffic monitoring, so when they find an illegal parking lot or an unauthorised vehicle, a small YOLO model can sound an alarm, take a picture at that time, accurately identify what has been found, where it is, and when, and then send this information to the central control and management system. Model compression technology will be employed to reduce the quantity of data transmitted per 24 hours of high-definition video and thus decrease the demand on cloud bandwidth.

1.3. Research motive and research framework

Some studies have been carried out on lightweight networking technology for edge computing, but there are still problems in how to balance the accuracy of the model with the compression ratio, how to allocate edge hardware resources, and how to build a layered architecture solution for cloud-edge-device. Therefore, this paper will provide a general overview of lightweight networking technology for the edge, introduce the basic ideas, current research status, problems, and future development directions of this technology, and offer relevant information to promote the construction of intelligent edge computing in the future. The organization of this paper is as follows: Section 2 presents the basic theories of neural networks; Section 3 introduces lightweight neural network technologies; Section 4 analyzes the deficiencies of the current technology and future development directions; finally, a general summary of this review is given in the conclusion.

2. Fundamentals of neural networks

A neural network is a mathematical model that imitates the connections among neurons in the human brain; it is a kind of large-scale computation model that consists of a very large number of

DOI: 10.54254/2755-2721/2026.CH37554

organised computational units in layers. Neural network models can learn features automatically, and then by adding more layers or combining different architectures, they can be used for all kinds of problems and tasks with different degrees of nonlinearity. As shown in Figure 1, the general working process of a neural network is as follows. Given a set of input values, the first operation of an artificial neuron is an affine transformation, which is a linear function.

The above is known as pre-activation values. Although a stack of many layers can be used, it is still only a single linear function and thus cannot model complex nonlinear functions. Therefore, an activation function is used to introduce nonlinearity. Therefore, deep learning is quite good at recognising and analysing information. As a single neuron can only produce a single number, and many real-world problems need several features, several independent neurons must cooperate to achieve this. The group of cooperative neurons is in a row. Concatenating all the neuron weights gives a weight matrix; concatenating all the biases yields a bias vector. A deep neural network has several layers.The output vector of the previous layer is used as the input for this layer.

In fact, a single-layer network is a group of neurons that can perform linear transformations and nonlinear fitting of images by means of matrix operations. To form a whole deep neural network, several layers are stacked one on top of the other and the output of one layer is used as the input for the next. The three general parts of a neural network are the input layer, the hidden layers and the output layer.

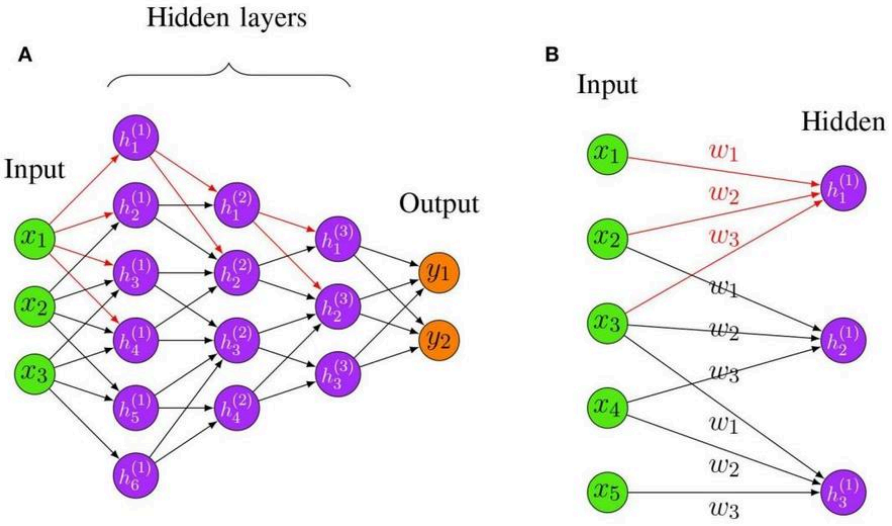


Figure 1. The simplest feedforward networks: (a) a shallow feedforward network with a depth of 2; (b) a deep feedforward network with a depth of 4 [6]

Network depth is the number of nonlinear transformations in a network, and it can be regarded as the number of layers of the network. It is employed to obtain characteristics of the network directly. Shallow Networks can only find the edges and lines of objects; Deep Networks can also find general features in images. Network width is the number of neurons in a hidden layer, and it is also the number of convolutional kernels at that level. The wider the layer, the more sub-features it can obtain and learn; thus, it is possible to obtain more detailed content features. However, it is also necessary to increase the number of model parameters and computational resources accordingly; thus, it will be resource-intensive. Therefore, this model cannot be carried out at the edge due to its small power and memory [6].

Convolutional Neural Networks (CNNs) are one of the typical kinds of deep learning models. AlexNet is a type of CNN that nearly halved the error rate in the 2012 ImageNet challenge; as a

DOI: 10.54254/2755-2721/2026.CH37554

result, it achieved excellent results in deep learning research and became one of the most influential papers in artificial intelligence. A Convolutional Neural Network is a kind of feedforward neural network. The main parts of a typical convolutional network are a sequence of fully connected layers, as well as convolutional layers, pooling layers and so on. It does not need to identify every pixel in the traditional network, so it will have a small computational cost and will not alter the original context of the image. Convolutional kernels in a Convolutional Neural Network are employed as weight matrices to extract features from images and thus reduce the number of parameters in image recognition. The larger the size of the convolutional kernel, the larger the area of pixels in the image that it covers during computation; thus, its receptive field is extended and it can gather more global information. The above examples are only of single-channel convolution operations, but in practice, images are generally in an RGB three-channel format. In such cases, the number of convolution kernels needs to be equal to the number of channels in the output feature map, and the depth of the convolution kernel itself must be set to the number of channels in the input feature map [6].

The pooling layer will discard some redundant parts of the feature map to keep only the important features, reduce the size of the feature map, and thus decrease both the computational cost and the number of parameters in the model. The above is also known as downsampling. Pooling layers are also kinds of sliding windows for feature map recognition, and you can manually set the size of the pooling kernel to adjust the recognition accuracy of the model. The two kinds of pooling are Max Pooling and Average Pooling. Max pooling takes the largest value in the pooling window as the output after moving the window one by one. As the model has only kept the most prominent features of the image, a small movement of the object during scanning will change the pooled output slightly. One of the reasons for learning in models is that they are translation-invariant; this is called translation invariance. Average pooling, on the other hand, takes the average of all values in a pooling kernel to get the output value and retains the general information of the data. Flatten the 2D feature map obtained after pooling, then convert it to a 1D vector and pass it to the fully connected layer. The first purpose of the fully connected layer is to gather the local features obtained in the previous layers and process them. Based on the features of colour and lines in an image, the Softmax function is used to compute the probability distribution for each class, and finally, the class with the highest probability is taken as the recognition result [6].

3. Current state of research on lightweight networking technologies

3.1. Pruning

The pruning algorithm finds the contribution value of each part in the network and then removes the connections that have a small contribution to reduce the number of parameters and computation (FLOPs). For example, in magnitude pruning, scores are based on the absolute values of the weights, and the parameters with the smallest absolute values are removed; this is the most basic form of pruning. Pruning algorithms are either static pruning or dynamic pruning.

Based on the level of pruning, the two kinds of static pruning are unstructured pruning (weight pruning) and structured pruning (neuron pruning) [7]. The smallest unit of unstructured pruning is a single weight parameter. In this way, one can determine the relative weights of all the items independently; unnecessary weights will be removed, and an irregular sparse weight matrix will be obtained to reduce the size of the model. However, the problem is that the pruning of useless weights is too scattered; thus, it is difficult for GPUs, NPUs and other edge AI chips to utilise this architecture for inference acceleration. Structured pruning is, on the other hand, based on an entire network module and removes whole channels directly to obtain a subnetwork with dense output

DOI: 10.54254/2755-2721/2026.CH37554

weights. It has good rules that are more suitable for speeding up inference on edge devices; although there will be a small drop in accuracy, it can be recovered by simple fine-tuning. For example, to reduce the number of channels, sparsity regularization can be added to the γ coefficients of the BN layer in the Network-Slimming method to obtain an organised output network. The first is a practical way to realise the hardware [7].

Both have strengths and weaknesses, but they are both considered static pruning methods. Some weights and channels are permanently removed after model training. Dynamic pruning is different from static pruning in that it can also temporarily hide model parameters, and based on changes in the input data during the operation of the model, it will decide whether to temporarily enable or disable some neurons or channels to maximise resource utilization [8].

3.2. Quantisation

Quantisation is a reduction in the precision of all or some of the parameters in a model by lowering the number of bits, and thus the storage requirements of the model are reduced. FP32 is an example; in this form, each weight is 32 bits long. Although this has a high degree of accuracy, it still requires a large amount of memory and will have high power consumption in the field. Quantisation can be employed to reduce the bit depth of 32-bit floating-point numbers to that of an 8-bit integer (INT8) in this case. Although the range of the values that can be represented is reduced and there is a small loss in precision, it is not significant for most engineering computer vision models. It can also be used for small models at the edge. The two main reasons for the reduction in model size are as follows: First, the weight distribution of large models is concentrated around zero, so low precision can be used to represent this high-weight region without losing much information. Second, there is the training process of stabilization. Because of natural noise in gradients, dropout is applied several times randomly to prevent overfitting. This is the main reason why quantization can still work even after an accuracy drop.

Based on the stage at which quantization is carried out, the two general types are post-training quantization and quantization-aware training. Post-training quantization is the direct quantization of a model's weights after the training has been carried out. A small amount of calibration data is needed to calculate the scaling factor S. It does not need to be retrained, has low debugging costs, and is fast. The problem is that there is a considerable reduction in accuracy when the number of resolution levels is reduced to four bits or less. Quantisation-aware training adds noise that mimics the noise of quantisation to help the model adapt to a low-precision environment. The accuracy of the trained model will be very high, but so will the computation cost [9].

3.3. Knowledge distillation

In knowledge distillation, the high-performing model is called the "teacher model" and the smaller models are called "student models". When the input sample is given to the teacher model, logits are output, and then a temperature-controlled softmax function is applied to obtain a smoothed probability distribution. The smoothed probability distribution not only contains the class label information but also has relationships among classes based on relative similarity; this hidden structure is referred to as "dark knowledge", and "dark knowledge" refers to the hidden information in the teacher model about the boundaries of similarity and uncertainty. The student model should learn the true labels in the dataset and use Kullback-Leibler (KL) divergence to make its output distribution approximate the smoothed probability distribution of the teacher's output. Knowledge

DOI: 10.54254/2755-2721/2026.CH37554

Distillation can be used to reduce the size of a model or to prevent an increase in error after pruning and quantisation.

The two types of distillation methods are black-box distillation and white-box distillation. The main difference is that the training model does not have access to the teacher model's internal parameters. In black-box distillation, only the prediction results of the teacher model are given to the student model, and its internal parameters cannot be directly accessed. White-box Distillation can be used to obtain the prediction results and parameters of the teacher model. In short, white-box distillation can be used to learn the parameters and knowledge representations inside the teacher model more efficiently.

Knowledge distillation can speed up learning, but it has many deficiencies in practice: the maximum performance of the student model is limited by the ability of the teacher model; if the teacher model is not very accurate, it is difficult for the student model to achieve good results; training the teacher model itself requires a large amount of computation and has high debugging costs; and fine-grained knowledge may be lost during the distillation process [10].

4. Limitations and prospects

4.1. Trade-off between model accuracy and compression ratio

Given the current situation, a high compression ratio and a very low bit rate are in conflict. Raising the compression level will reduce the amount of information the model can store. A drop in accuracy is a general problem of all model compression techniques. For example, the old pruning method has a relatively high compression rate, but regular quantization is not very accurate. Second, most of the current methods are symmetric and uniform quantization; if they are applied to networks with asymmetrically distributed weights, the accuracy of the network will be severely degraded [11]. Pruning and Quantisation will reduce the model's accuracy. At a high processing rate, the error will still be relatively small; otherwise, it will be too large [7, 9].

By improving the weighted optimisation pruning algorithm or identify models of the same type in the future, it will not have to solve the problem again every time switching networks, and thus increase the algorithm's applicability and versatility. Quantisers for the case of asymmetric data distribution can also be developed to extend the application range of the algorithm and improve the trade-off among accuracy and other factors in the modified model.

4.2. Difficulty of deployment in edge hardware

Edge GPUs have a high power consumption and are expensive, so deep neural network frameworks cannot be run in software, and compatibility is not good. At present, the toolchains in the industry need to be adapted by vendors beforehand. The pre-trained model will probably fail and cannot be used at the edge. Second, the instructions and software compilation stacks for NPUs from different companies are not the same; thus, there is fragmentation and an increase in the difficulty of debugging deployment problems on edge devices. In the future, extending the whole workflow of pruning to quantization and model compression and deployment will support a larger number of NPU hardware with different architectures and improve the cross-platform portability of the solution. It will continue to improve the entire software environment for NPU deployment and increase the compiler's recognition capability for new network models to reduce the amount of manual work in operator debugging and model conversion. Hardware-Software Co-optimisation Strategies for Compression Can Also Be Used. It will promote the integration of algorithms and

DOI: 10.54254/2755-2721/2026.CH37554

hardware to address the current problem that algorithms and hardware are designed independently of each other [12, 13].

4.3. Limitation in actual engineering application

TOPS is a representative kind of peak performance index. These are only theoretical numbers under ideal circumstances and do not reflect the actual operation results. For instance, the utilisation rate of the hardware MAC array will vary depending on the network layer and feature map size in different networks on the same TinyNPU. A chip has high peak computing power, but if the type of model layer does not match the computation pattern for which the hardware is well-suited, then the utilisation of hardware resources will be relatively low. Therefore, the TOPS values shown in the marketing materials for chips are not very representative. Micro-benchmarking has been carried out on the good-weather results to find the top indices. As the above indicators do not consider the actual usage conditions of the compute unit and memory contention, there is often a large gap between the listed specifications and the actual operating performance. The computer array is in addition to the chip specifications; it does not cover issues such as thermal throttling, the overhead of CPU-NPU interaction, or pre- and post-processing time. Therefore, the theoretical specifications may not be in line with the actual operating conditions of the embedded system. In the future, it will not only aim for high performance but also conduct all-encompassing model evaluation, monitoring the Mean Hardware Utilisation of all Network Layers. A fairer and more all-encompassing index for the assessment of edge accelerators should build. Reduce the weight of the peak TOPS in the hardware comparison. Based on the above hardware-aware compilation methods, it can reduce the waste of resources caused by calculation under different rules and further close the performance gap between the theoretical hardware metrics and the actual performance [12, 13].

5. Conclusion

In conclusion, Organise systematically the review of lightweight network technologies for edge computing in this paper, paying attention to pruning, quantization and knowledge distillation, which are the three main model compression methods, and conclude that generally, these compression methods are based on restructuring the network architecture and modifying pre-trained weights. Although considerable research progress has been achieved in the field of lightweight network technology, there are still three main bottlenecks in its application. In the future, it will need to continue to improve pruning algorithms and enhance their applicability and versatility to strike a good balance between model performance and compression ratio. At the same time, the existing pruning-and-quantization-based model compression deployment workflow needs to be extended to support a larger number of CPU architectures, and research on collaborative software-hardware compression strategies should continue to be carried out to promote the seamless integration of algorithms and hardware. Finally, a simple and effective way to evaluate light models at the edge should be introduced to reduce the demand for high-performance operation. This paper will offer some reference material for the research on lightweight networking technology in edge computing.

References

[1] Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine, 35*, 126–136.

[2] Xu, X., Zhou, Z., Hu, Z., Lin, S., & Yu, Z. (2024). Lightweight deep neural network models for edge intelligence: A survey. *Computer Science, 51*, 257–271.

DOI: 10.54254/2755-2721/2026.CH37554

[3] Bi, H. (2026). Lightweight model inference techniques in edge computing environments: Review. *Applied and Computational Engineering, 256*, 1–6.
[4] Liu, H., Galindo, M., Xie, H., Wong, L., Shuai, H., Li, Y., & Cheng, W. (2024). Lightweight deep learning for resource-constrained environments: A survey. arXiv.
[5] Ge, D., Li, H., Zhang, L., Liu, R., Shen, P., & Miao, Q. (2020). Survey of lightweight neural network. *Journal of Software, 31*, 2627–2653.
[6] Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., & Dehmer, M. (2020). An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence, 3*, 4.
[7] Wang, L., Song, Q., Geng, S., & Luan, Z. (2026). Survey of neural network filter pruning technology. *Computer Engineering and Applications, 62*, 1-25.
[8] Zhang, M., Lu, Q., Li, W., & Song, H. (2021). Deep neural network compression algorithm based on combined dynamic pruning. *Journal of Computer Applications, 41*, 1589-1596.
[9] Huang, X., Qu, Z., Ye, B., Hu, S., Jia, N., & Qian, L. (2026). Learning-driven model quantization: Methods, challenges and prospects. *Journal of Computer Research and Development, 63*, 1950-1987.
[10] Wu, Z., Sun, S., Wang, Y., Liu, M., Jiang, X., Li, R., & Gao, B. (2023). Knowledge distillation in federated edge learning: A survey. arXiv.
[11] Ning, X., Zhao, W., Zong, Y., Zhang, Y., Chen, H., Zhou, Q., & Ma, J. (2024). An overview of the joint optimization method for neural network compression. *CAAI Transactions on Intelligent Systems, 19*, 36–57.
[12] Zeng, H. (2025). Research on NPU hardware acceleration technology for convolutional neural networks based on pruning and quantization. Southwest University of Science and Technology.
[13] Xu, K. (2022). Research on design of energy-efficient deep neural network acceleration chip. Xidian University.