Mechatronics and Automation Technology
J. Xu (Ed.)

doi:10.3233/ATDE221161

A General Neural Network Deployment Framework for Edge Devices

Jianshan WANG[a], Yi WANG[b], Wange LOU[c] Wenxia WEI[a,1] and Jiaxian LV[d]
[a] *College of Information and Communication, National University of Defense Technology, Wuhan, Hubei, China*
[b] *Military Vocational Education Center, National University of Defense Technology, Changsha, Hunan, China*
[c] *State Key Laboratory of Virology, College of Life Sciences, Wuhan University, Wuhan, Hubei, China*
[d] *PLA Information Security Test and Evaluation Center, Beijing, China*

Abstract. As the expansion of edge devices, such as local edge servers or even IoT (Internet of Things) devices, edge computing reforms data processing flow, engendering improvement in faster insights, improved response times and better bandwidth availability. However, industry encounters computational resource constraints of machine learning on edge devices, e.g., limitation of memory and processing power. To tackle this situation, we attempt to enhance flexibility of neural network models for variable hardware constraints. In this paper, we propose a paradigm of neural network combination implementation as a primitive solution; further more, we manage to build up a general neural network deployment framework for edge devices.

Keywords. Edge computing, neural network, tinyML.

1. Introduction

As the expansion of distributed devices, the information expressway towards central data centers becomes sluggish stream because of the data explosion from the edge of the network. Edge computing, as a distributed computing framework, reforms data processing flow so that domain specific applications turn on closer to data receivers such as local edge servers or even IoT devices. Considered as an evolution of cloud computing[1], the proximity to data at its source is able to engender improvement of information processing, including faster insights, improved response times and better bandwidth availability.

Edge devices, or the massive distribution of low-power chips, exposed to more peripheral perception, become operation platforms of originally energy-intensive artificial intelligent applications, as is depicted in figure 1. These devices are supposed to be deployed in any place without much maintenance.

Naturally, researchers start to introduce machine learning models into edge devices. Since Google launch the TensorFlow Lite in 2017, tinyML[2] has penetrated into the

[1] Corresponding Author, Wenxia WEI, College of Information and Communication, National University of Defense Technology, Wuhan, Hubei, China; E-mail: weiwenxia17@nudt.edu.cn.

world of IoT, bringing sophisticated neural network models into edge computing. While at the same time, leading the way before data center into traffic jam. Raw data transmission from device itself to high performance unit is realistically too power-consuming to be a reasonable approach. Generating actionable information locally under restricted processing resource becomes a prudent choice.

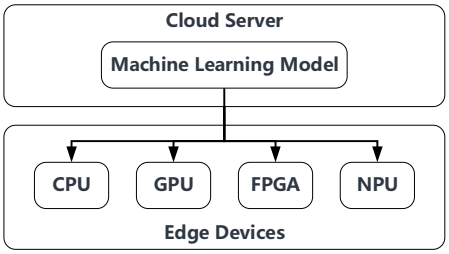

Figure 1. Machine learning model is trained on cloud server and deployed on edge devices.

To deploy machine learning applications on edge devices, researchers from academia have to deal with mussy runtimes and variable frameworks. Thankfully, contributors from communities build up an open ecosystem for interoperable AI models, which makes it simple and efficiency to transport neural network models among platforms. Open Neural Network Exchange (ONNX)[3] is an open format for representing machine learning models. In our experiments, the neural network is abstracted into a scalable computational graph model provided by ONNX, and then refactored on a specific edge device.

In this paper, to tackle the situation of limited computational resources, we attempt to enhance flexibility of neural network models for variable hardware constraints. In this paper, we propose a paradigm of neural network combination implementation as a primitive solution; further more, we manage to build up a general neural network deployment framework for edge devices.

2. Combinational Neural Network

Edge computing is the essential solution to the excessively centralized system. This paradigm tends to increase participation of data processing closer to where it is generated, mitigating data throughput pressure over the network. Confronted with boosting diversity of edge devices, we propose combinational neural network as a primitive exploration of hardware-adapted neural network.

A combinational neural network (coNN) is the basic model which can generate sub-networks according to different architectural configurations. Each sub-network is supposed to maintain competitive performance in accuracy and latency compared to the original intact network. Contrary to traditional convolutional neural network (CNN) of stable hierarchies and connections, coNN struggles to achieve consistent potency under arbitrary hardware constraints through architectural recombination and deletion. To be specific, coNN is able to downsize its network, e.g., depth, width and kernel size, to match limited computational resources and storage memory of a given chip.

2.1. Architectural Design Principle

A coNN model is a fully functional CNN model essentially and is able to deployed directly on unbounded computational platform. In order to forge a coNN model with flexible architecture of different computing resource requirements, researchers should follow the architectural design principles of coNN below.

- Alterable resolution. The coNN model should be capable of incepting random and variable sensor data depending on the scenario.
- Alterable depth. The coNN offers alterable scale of neural networks layers to construct sub-networks consistent to the storage limitation.
- Alterable width. The coNN allows impact ranking of channels between layers and imposes appropriate deletions to generate sub-networks.
- Alterable kernel size. The coNN offers arbitrary kernel sizes.

Follow the principles above, researchers are convenient to design their own coNN model, matching specific scenario requirements. To illustrate real implementation of these design principles, we take a modification of CNN model as our example. In this case, the input picture size ranges from 144 to 256, which realizes alterable resolution instinctively; the model is divided into slices from from to back, each slice comprises several layers and the quantity of layers is chosen from {1, 2, 3}, which realizes alterable depth; the width, number of channels between layers, is decided by the compression ration chosen from {1, 2, 4}, which realizes alterable width; the kernel size ranges in {1, 3, 5, 7}, which realizes alterable kernel size.

2.2. Training Paradigm

Following practice of CNN models, we propose a training paradigm of coNN implementation. As long as researchers have the experience of building up CNN models, it is fast-paced to adopt a few modifications to embrace coNN. To put it bluntly, researchers could apply a naïve and primary training paradigm, enumerating all sub-networks of coNN and optimizing one by one from scratch. In this case, the number of sub-networks is the linearly decisive factor of training time complexity. Unfortunately, when the scale of coNN increases, the number of sub-networks boosts up exponentially. Thus, we have to introduce some tricky approaches to avoid computational explosion.

We suggest researchers to enforce itemized training paradigm to traverse forest of coNN and its sub-networks. For example, after training the whole basic coNN, we alter the architectural setting of alterable depth, alterable width, alterable kernel size, and train the coNN under each setting step by step.

3. Framework

The coNN is the core of our deployment framework for edge devices.

When we are confronted with specific business scenarios, we analyze the scenario requirements and design the corresponding combinational neural network. After the network specifications are submitted to the cloud server, the model will be trained on mainstream machine learning platforms like PyTorch, with alterable architectural configurations. As accuracy of the model reaches stable and reliable level, it will be

exported as combinational ONNX model, which is convenient to be deployed on variable hardware platform. The deployment framework is depicted as figure 2.

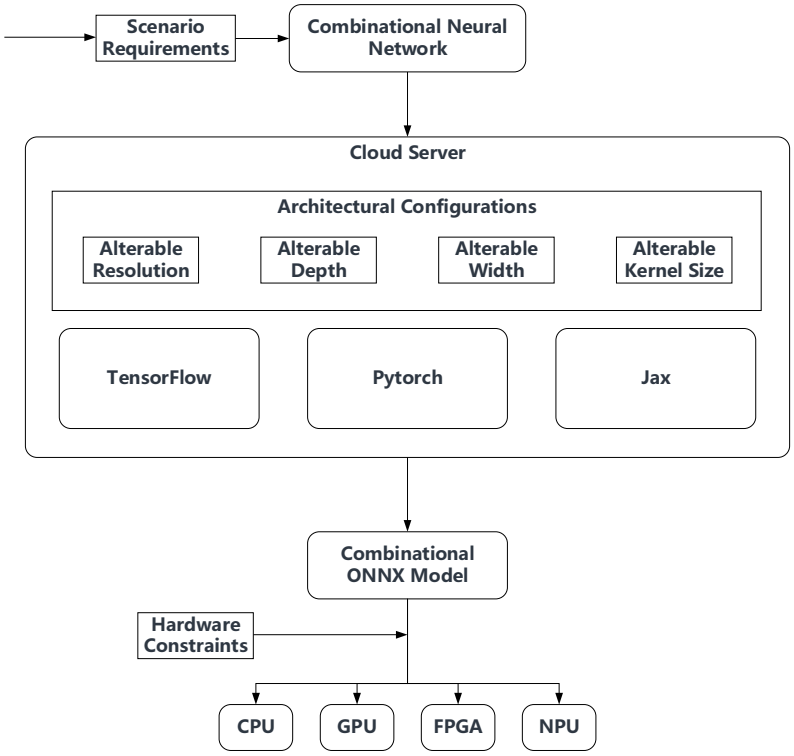

Figure 2. Neural network deployment framework.

3.1. Model Adaptation and Testing

Combinational ONNX models should be modified according to the hardware constraints. A naïve implementation of this requirement is to deploy the minimum model and increase model scale until emergence of performance degrading.

Table 1. Compared with AlexNet, our prototypical implementations of combinational neural network, including original version and compression of 50% and 25%, achieve similar accuracy on ImageNet[4] with smaller model.

Model	Size	Top-1 ImageNet Accuracy
AlexNet(baseline)	240.3MB	57.2%
coNN(original)	48.6MB	59.8%
coNN(50%)	23.7MB	58.2%
coNN(25%)	10.7MB	57.1%

We explore our primitive prototype on Qualcomm Snapdragon 810 with 3GB RAM. Compared with the AlexNet[5] as the baseline, our prototypical implementation of combinational neural network achieves stable performance with less computational resources, as showed in table 1.

3.2. Framework Compatibility

The compatibility of our deployment framework is contingent on the versatility of ONNX. Open Neural Network Exchange (ONNX)[3] is an open format built to represent machine learning models. In our experiments, neural networks are abstracted as an an extensible computation graph model provided by ONNX and then reconstructed on specific edge device.

4. Related Works

There are many profound explorations of applicating machine learning models on edge computing platforms. Researchers tend to carry forward in two directions: first, improving the hardware efficiency of running a given neural network; second, fine-tuning machine learning models by automating the architecture design process or proposing neural networks with more flexible and dynamic architecture.

4.1. Machine Learning Framework for Low Power Platform

The most famous and prevailing platform for low power platform is TinyML[2] and NCNN[6].

TinyML, based on the consensus made across industry and academia, is a technical framework of machine learning model translation, especially constructing energy-efficient neural network model for the time being.

NCNN is an inference computing framework optimized for mobile platforms, supporting most deep learning models deployment of mainstream framework.

At present, platforms like the Raspberry Pi require external power supply or built-in battery like smartphone. Thus, we could not take it as the typical platform deploying TinyML model. But with the convenience of frameworks like TensorFlow, these platforms are intuitionistic testbed of our improvement.

4.2. Neural Network Architecture Design

Researchers are zealous to design ingenious architecture of neural networks to operate on specific hardware configurations appropriately and efficiently. For example, SqueezeNet[7] applies replacement of smaller filters and less channels to support feasible FPGA and embedded deployment; MobileNets[8] introduces model shrinking hyperparameters to build lightweight deep neural networks for mobile and embedded vision applications. Shufflenet[9] considers the direct metric, such as speed, instead of indirect metric like FLOPs, to conclude empirical guidelines of efficient network architecture.

Most efficient neural networks adopt network pruning, while model compression starts to make a difference. Trained Ternary Quantization[10] intends to reduce the scale of the neural networks through degrading the precision of network weights into ternary values.

5. Conclusions

Edge computing is a complement of centralized computing for users to act on data at the source. The massive distribution of low-power chips on Android and IOS devices, or even manufacture line micro-controller units, are confronted by the noisy sensor data, which is impossible to analyze and utilize efficiently.

However, industry encounters computational resource constraints of machine learning on edge devices, e.g., limitation of memory and processing power, especially only a limited number of feature channels is affordable under a given computation budget. To tackle this situation, we attempt to enhance flexibility of neural network models for variable hardware constraints. In this paper, we propose a paradigm of neural network combination implementation as a primitive solution; further more, we manage to build up a general neural network deployment framework for edge devices.

References

[1] Chen J and Ran X, "Deep Learning With Edge Computing: A Review," Proc. IEEE, 2019.
[2] "GitHub - mit-han-lab/tinyml." [Online]. Available: https://github.com/mit-han-lab/tinyml. [Accessed: 27-Jun-2022].
[3] "GitHub - onnx/onnx: Open standard for machine learning interoperability." [Online]. Available: https://github.com/onnx/onnx. [Accessed: 27-Jun-2022].
[4] K. Yang, K. Qinami, L. Fei-Fei, J. Deng, and O. Russakovsky, "Towards fairer datasets: Filtering and balancing the distribution of the people subtree in the ImageNet hierarchy," FAT* 2020 - Proc. 2020 Conf. Fairness, Accountability, Transpar., pp. 547–558, Jan. 2020.
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks."
[6] "GitHub - Tencent/ncnn: ncnn is a high-performance neural network inference framework optimized for the mobile platform." [Online]. Available: https://github.com/Tencent/ncnn. [Accessed: 27-Jun-2022].
[7] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016.
[8] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications."
[9] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design."
[10] C. Zhu, S. Han, H. Mao, and W. J. Dally, "TRAINED TERNARY QUANTIZATION."