Computers, Materials & Continua CMC, vol.65, no.3, pp.2321-2334, 2020

MoTransFrame: Model Transfer Framework for CNNs on Low-Resource Edge Computing Node

Panyu Liu[1], Huilin Ren[2], Xiaojun Shi[3], Yangyang Li[4, *], Zhiping Cai[1], Fang Liu[5] and Huacheng Zeng[6]

Abstract: Deep learning technology has been widely used in computer vision, speech recognition, natural language processing, and other related fields. The deep learning algorithm has high precision and high reliability. However, the lack of resources in the edge terminal equipment makes it difficult to run deep learning algorithms that require more memory and computing power. In this paper, we propose MoTransFrame, a general model processing framework for deep learning models. Instead of designing a model compression algorithm with a high compression ratio, MoTransFrame can transplant popular convolutional neural networks models to resources-starved edge devices promptly and accurately. By the integration method, Deep learning models can be converted into portable projects for Arduino, a typical edge device with limited resources. Our experiments show that MoTransFrame has good adaptability in edge devices with limited memories. It is more flexible than other model transplantation methods. It can keep a small loss of model accuracy when the number of parameters is compressed by tens of times. At the same time, the computational resources needed in the reasoning process are less than what the edge node could handle.

Keywords: Edge computing, convolutional neural network, model transformation, model compression.

1 Introduction

As a hot spot in artificial intelligence, deep learning technology has been sought after by academics and industries in recent years. Deep learning technology has been focusing on computer vision, natural language processing, speech recognition, and bringing about new related products [Cheng, Wang, Zhou et al. (2017)]. Since deep learning training and

[1] National University of Defense Technology, Changsha, 410073, China.

[2] Training and Administration Department, the Central Military Commission, Beijing, 100851, China.

[3] Department of Science and Technology, China Electronics Technology Group Corporation, Beijing, 100846, China.

[4] National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data, Beijing, 100041, China.

[5] School of Design, Hunan University, Changsha, 410082, China.

[6] Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292, USA.

[*] Corresponding Author: Yangyang Li. Email: liyangyang@cetc.com.cn.

Received: 08 March 2020; Accepted: 17 July 2020.

CMC. doi:10.32604/cmc.2020.010522 www.techscience.com/journal/cmc

reasoning need a lot of computation power [Li, Ota and Dong (2018)], artificial intelligence applications based on deep learning usually appears in the powerful cloud computing data center [Liu, Tang, Li et al. (2019)].

The edge terminal equipment with limited resources is ubiquitous in various application scenarios. Its popularity has attracted extensive attention from the academic and industry to deploy deep learning models in resource-constrained [Chen, Yu, Liu et al. (2019)] terminal devices to make intelligent applications more accessible to users. To improve models reasoning efficiency, edge computing intelligence technology [Guo, Liu, Xiao et al. (2019)] integrates the complementary advantages of local inference and server computing power [Liu, Cai, Xu et al. (2015)]. It achieves the purpose of significantly reducing the delay and energy consumption [Wu, Zhang, Cai et al. (2020)] in deep learning model reasoning.

Edge computing is of current importance. One manifestation is that many large companies have invested billions of dollars in edge computing technology. In 2008, the number of edge terminals which were officially announced, exceeded the total number of living humans. It is estimated the number of edge devices will reach 50 billion by 2020. However, combining edge computing and machine learning has brought new experiences, new opportunities, and new challenges to smart homes [Cai, He, Sun et al. (2017)], smart factories, and smart city industries. In addition, that combination can enable the intelligent edge device to perform calculations and inferences locally. Thus, it helps terminal saving time to transfer data to the cloud, protecting the data privacy, reducing network traffic [Zhang, Lu, Li et al. (2019)].

Running deep learning on edge nodes can provide services and process tasks more quickly. The edge here refers to consumer products that perform calculations locally. Although popular deep learning models have achieved the best accuracy in many classification and prediction applications [Zhang, Wang, Lu et al. (2019)], CNN's reasoning for edge nodes is still challenging due to severe limits of computing resource. Deep Learning models are known for their large sizes and high computational costs [Li, Zhou and Chen (2018)]. Arduino series development boards have very constrained resources; Tab.1 compares Arduino with similar IoT processors. The broad proliferation of MCUs relative to desktop GPUs and CPUs stems from the fact they are cheaper and consume several orders of less powerful than desktop GPUs and CPUs.

Table 1: Processors for ML inference: Performance comparison of similar chips

Edge Platform	Processor	Type	SRAM	Flash	Power	Frequency	Price
Raspberry Pi 2B	ARM Cortex-A7	IoT	1 GB	1GB+	1.20 W	900 MHz	$38.00
STM32 F723	ARM Cortex-M7	IoT	256 KB	512 KB	170 mW	216 MHz	$11.85
FRDM-K64F	ARM Cortex-M4	IoT	256 KB	1 MB	58 mW	120 MHz	$35.00
Micro:Bit	ARM Cortex-M0	IoT	16 KB	256 KB	18 mW	16 MHz	$13.21
Arduino MKR1000	ARM Cortex-M0	IoT	32 KB	256 KB	14 mW	48 MHz	$37.40
Arduino Mega	ATmega2560	IoT	8 KB	256 KB	~1 mW	16 MHz	$28.00
Arduino Micro	ATmega32u4	IoT	2.5 KB	32 KB	~1 mW	16 MHz	$18.80
Arduino Uno	ATmega328P	IoT	2 KB	32 KB	~1 mW	16 MHz	$11.86

The key problem of edge intelligence is how to deploy large deep learning models in the resource-constrained edge node, including deep learning model's compression, models transplantation [Rastegari, Ordonez, Redmon et al. (2016)], and collaborative scheduling between edge servers and terminal devices.

The general deep learning model is composed of multiple types of networks layers. Since the demand for computing resource and the amount of output data is different in various network layers, a simplistic idea is to divide the entire deep learning model into two parts, in which a large part is divided into edge servers for calculation, as shown in Fig. 1, and a small portion of the calculations that require fewer resources runs on the local node [Teng, Ota, Liu et al. (2020)]. It can reduce the inference delay of the deep learning model by collaborating cloud and edge [Huang, Zhang, Zeng et al. (2020)]. However, choosing different model partition points will lead to different calculation times.

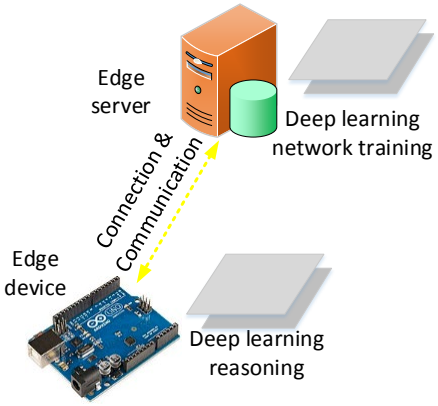


Figure 1: Portion of the calculation is kept locally in the terminal device

Choosing an inappropriate segmentation point may result in an edge node being unable to perform calculations due to limited resources, such as the several tasks which need more calculations are kept in the computation center and thus will increase network latency [Li, Wang and Kong (2018)]. The objective of this research is to answer the following questions:

(a) How to transplant a deep learning model to the edge node?

(b) Is it feasible to run a deep learning model for a resource-scarce edge node?

(c) What measures can be taken to transplant the deep learning model to the edge node easily and quickly?

(d) What is the next step needed to apply deep learning to edge computing?

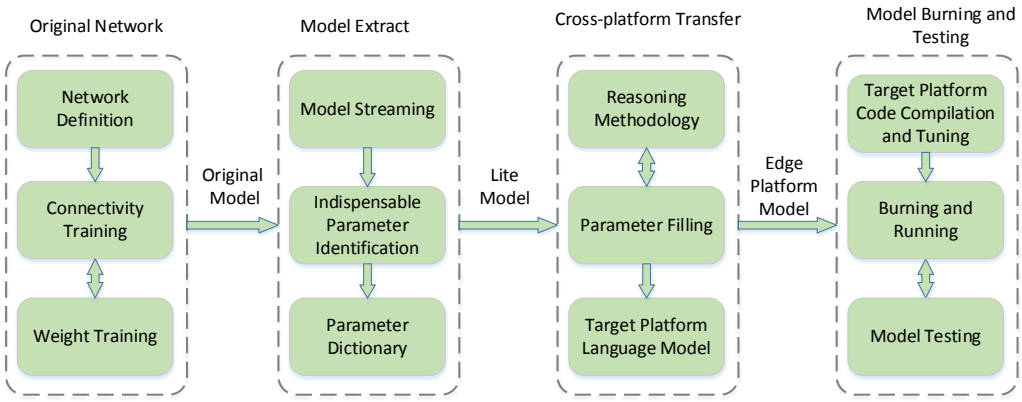


Figure 2: Four-stage pipeline for Model Transfer Framework

Our goal is to transplant small and medium-sized convolutional neural network models to low-resources edge devices directly, thus, the inference can be performed on the edge device locally. To meet this end, we propose a model transformation framework: a four-stage pipeline as shown in Fig. 2. This framework can not only realize the rapid transplantation of the model in the terminal nodes, but also significantly reduce the memory usage during the model reasoning locally. Firstly, the network structure is adapted or trimmed to a lightweight during model training. Several fine-tuning methods of deep learning training can be performed at this stage to maximize the model accuracy. Secondly, only the important part of different type of features are extracted after the model is already trained and saved in the first stage, thus, only the necessary parameters in the already trained model are saved. Some variables, such as environmental variables, are not necessary parameters [Yu, Cai, Wang et al. (2019)] for model reasoning, we do not need to keep them. The redundant part of the model variables can be removed as much as possible. Thirdly, the inference rules of the deep learning classification algorithm are analyzed. Fill in the extracted parameters according to reasoning rules and use automatic coding to generate Arduino project files i.e., the transformed model. Fourthly, the Arduino project can be compiled and run on the edge embedded platform directly, such as, Arduino mega.

The rest of this paper is organized as follows: Section 2 discusses the related work of the optimization of deep learning model. Section 3 introduces the process and methodology of the model transfer framework proposed and the experimental work. Section 4 presents the results and observations from the experiments. Section 5 presents the conclusions.

2 Related work

Recent years, we have seen extensive study of designing a fitting CNNs model to fit the low-resources edge node. According to the computing power of edge nodes, edge hosts for model transplantation are classified into three different categories: The first is the mobile devices with reliable strong performance, such as mobile phones. There is also a kind of large-scale mobile electric vehicle (EVs) [Tang, Wang, Song et al. (2019)] which can be included in the first category. The second is the embedded devices with relatively

strong performance, such as Raspberry Pi. The last is the small development boards with weak performance, such as Arduino Mega or Arduino UNO R3. The model training optimization methods such as network pruning [Tseng, Bhattachara, Fernández-Marqués et al. (2018)], parameter sharing, and parameter quantization can be implemented on all three types device. Chowdhery et al. [Chowdhery, Warden, Shlens et al. (2019)] used network pruning technology. During the model training, the parameters below zero are replaced by zero. However, this caused a loss of accuracy. Han et al. [Han, Mao and Dally (2015)] further compressed the model size by integrated three technologies: network pruning technology, parameter quantization technology and Hoffman code. They call this deep compression technology. Besides, the team designed a hardware accelerator called EIE. This hardware accelerator can directly run the compressed model to improve the running speed and reduce memory consumption. Han's deep compression technology [Han, Kang, Mao et al. (2017)] has a good compression ratio for large network models. For example, it can compress AlexNet by as much as 35 times and has the least impact on accuracy at the same time. However, a deep compression method can cause a significant accuracy loss in small network structure, so this method is not feasible in model adaptation for low-resource edge nodes.

The image recognition optimized model proposed by Liu et al. [Liu, Cao, Luo et al. (2017)] is utilized to divide the image recognition process into two layers: building the edge layer locally on the mobile device and building a remote service in the cloud. Courbariaux et al. [Courbariaux, Hubara, Soudry et al. (2016)] proposed a software accelerator: DeepX. DeepX can significantly reduce equipment resources usage, which is a severe bottleneck for large-scale mobile applications. Li et al. [Li, Wang and Kong (2018)] designed an acceleration framework named DeepRebirth by "slimming" the existing continuous parallel non-tensor layers and tensor layers. Yazici et al. [Yazici, Basurra and Gaber (2018)] presented the feasibility of running several ubiquitous machine learning algorithms on IoT edge devices. Kumar et al. [Kumar, Goyal and Varma (2017)] proposed Bonsai to cut the model size and prediction cost while preserving prediction accuracy. Yet, the model cannot be transplanted to edge nodes quickly and easily.

3 MoTransFrame

Due to the strong dependence of model training on the deep learning framework, it is impossible to transplant the model directly to the edge nodes, but manual transplantation is too costly. Thus, it is necessary to design a general transplantation framework to automate the model transplantation. According to the process and rules of model transplantation, we designed a good performance of CNNs model transplantation framework. We denote the already trained model as an weights matrix object, which is expressed as $\omega=\{weights, bias\}$, among them: $weights=\{conv, FC\}$ describe the network convolution weights; The *conv* represents different convolution layer weights; *FC* represents the weight of the network full connection layer; *Bias* represents weight bias in each layer of the network. Our overarching goal is to identify the CNNs model architecture while maintaining competitive accuracy. To achieve this goal, we adopt three main strategies when designing a model transfer framework:

Strategy 1. Without designing a unique network structure, the original one is lightly pruned, and the accuracy of the original model is retained to the greatest extent. Given an already trained model, we will extract the corresponding weight according to the structure of the network computing graph. The structure of the model can be obtained by the structure defined in the training process.

Strategy 2. Input the trained model into the model interpreter framework we designed. Consider a convolution layer consisting of 5×5 filters. The total number of parameters in this layer is (number of input channels)×(number of filters)×(5×5). The framework reorganizes parameters to generate the matching embedded language code format according to the deep learning inference rules. It not only reduces model inference's dependence on the environment but also cuts out unnecessary parts of the original model.

Strategy 3. After the model is transformed into the Arduino language version, the model files are burned into the edge node. One of the most significant advantages is the converted edge version of the model files can be directly compiled and run on the Arduino platform. The converted model can then be tested to record the accuracy of the test data set and record the runtime energy consumption of each instance.

3.1 Model extraction

Model parameters are proposed to be extracted according to the parameters needed for deep learning classification. Eq. (1) describes the calculation process of the usual CNN classification model reasoning. According to the formula, the inference calculation of the classifier model can be realized by matrix product of each layer.

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}^T \underbrace{kernel[0]}_{n_i \times 5 \times 5} + \underbrace{bias[0]}_{1 \times n_i} \xrightarrow[Pooling]{ReLU} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix}^T \underbrace{kernel[H-1]}_{n_{i+1} \times p_{H-1}} + \underbrace{bias[H-1]}_{1 \times n_{i+1}}$$

$$\cdots \xrightarrow[Pooling]{\mathrm{Re}lu} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_M \end{bmatrix}^T \underbrace{FC[0]}_{n_j \times M} \xrightarrow{View} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n_j} \end{bmatrix}^T \cdots \xrightarrow[k]{argmax} y_k \tag{1}$$

Eq. (1) depicts that the calculation patterns of convolution, activation and pooling at each layer are similar. We use this feature to propose a general parameter extraction method. We read the model in the way of data flow. Generally, the model file is in the form of a dictionary. The key of the dictionary is the variable name of the parameter, and the value of the key is the value corresponding to the variable name. According to this formula, we can clearly know which variables need to be extracted, and then use these variables as keys to retrieve the model file to obtain the parameter matrix. Our model extraction method is intended to collect CNN model with different type of parameters. Each trained model consists of fixed parameter blocks and variable parameter blocks, and each variable parameter block contains a mandatory parameter layer. We consider mandatory parameters is necessary for model reasoning. By extracting the original model parameters

accurately, we can minimize the loss of model accuracy.

3.2 Conforming memory requirements

Since the deep learning model is dependent on TensorFlow, PyTorch, Caffe, and third-party imported libraries, it is impractical to port all related libraries into the Arduino memory. In this case, training a deep learning model that can run the inference logic in the edge node with only 8KB RAM directly is very crucial. We have to consider the usage of the edge node memory carefully. Our goal is to meet the storage requirements of Arduino when designing the training network structure, so the trained model can be directly converted by the transformation framework. Optimization methods for model size includes the following four steps:

1. Calculate the size of the deep learning network structure and the trained model;
2. Extract the parameters from the trained model according to the network structure and calculate the size of the parameters.
3. Obtain the maximum data size allowed by the target device flash memory, and calculate the space size needed for the storage of the converted cross-platform model;
4. If the size of the converted model meets the target device storage requirements, the model can be directly transplanted to the edge node for compilation and running. Otherwise, repeat the first step. In the last case, adjusting the size of the original model by changing the neural network structure to meet the limit of the edge node's memory.

3.3 Weights quantization and model adaption

Based on the method of network pruning in model compression, different thresholds are set in the training process to determine whether a connection needs to prune. Different levels of pruning can obtain different sizes of deep neural network models, and it can further reduce the size of the model through parameter quantification [Zhang, Wang, Li et al. (2018)]. Implementation of quantization is achieved by converting common operations into equivalent octet versions. These operations involved convolution, activation function, pooling, and splicing. For example, mapping a 16-bit floating-point number to an 8-bit integer can reduce the model size by 4 times. We find that the parameter values of the same layer of the model distributed in smaller intervals, and the distribution of the parameters conform to the normal distribution law. We record the minimum and maximum values. In the 8-bit quantization (together with other options), we map all parameters of the same layer linearly (or use nonlinear mapping to compress the space further) to an appropriate interval.

The interval [min, max] is evenly divided into 255 cells, and all input values correspond to the interval values. In this way, the size of the model is compressed into 25% compared to the original model size. All parameters can be restored when the weights are loaded into the Arduino memory.

It has been proved that if the model uses 8-bit integer parameters in the reasoning process. The calculation speed of the inference will be improved greatly. The reason is that the gradient value has to be calculated in the training process, and the parameters are continuously updated according to the loss function. Each time the parameter changes are

microscopic. Therefore, the training process of the model requires high-precision floating-point values. In the process of model reasoning, we also need to use the integer value instead. We note that the output is still represented by a floating-point number. The intermediate calculation is always an 8-bit number.

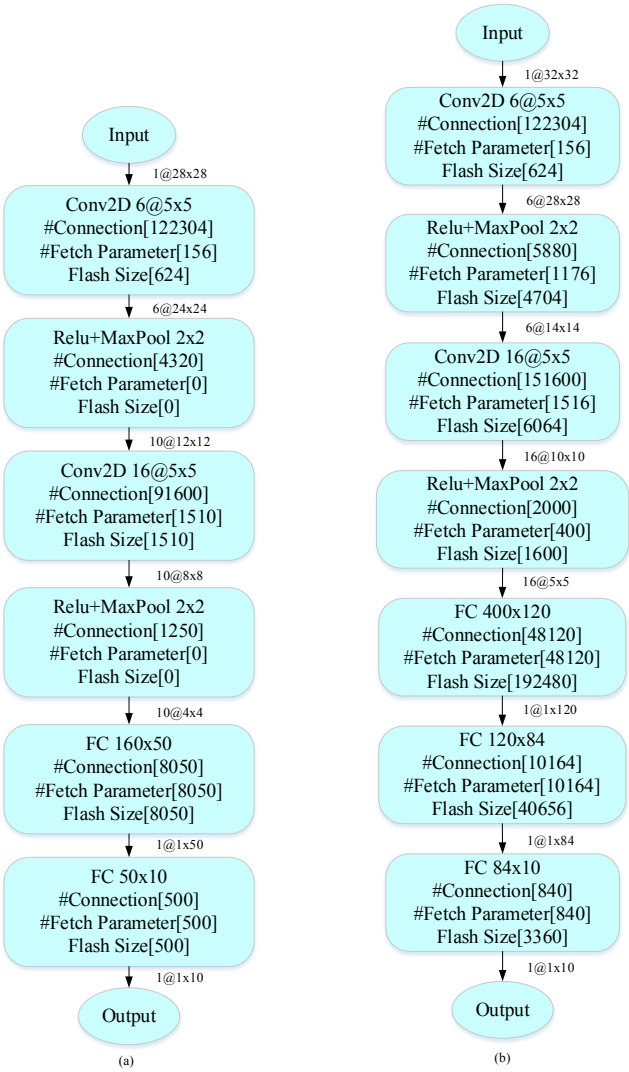


Figure 3: Model architectures found with the best test accuracy on MNIST-10(a), and compared the original LeNet-5 structure(b), each node in the graph are annotated with connection, parameter and flash size, # represents the number of corresponding objects, and the values in square brackets show the quantities

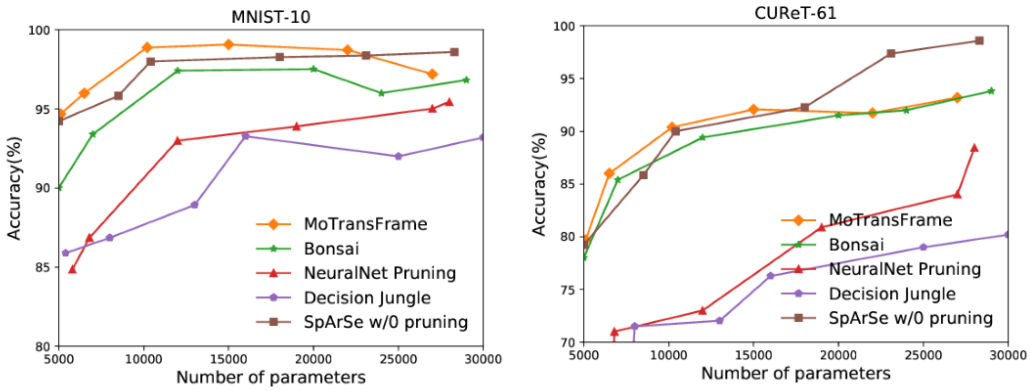
4 Results and discussions

LeNet-5 is a classic CNN classification architecture, which consists of two convolutional and average pooling layers, followed by a flattening convolutional layer, then two full

connection layers, and finally, a SoftMax classifier. We find that the size of the middle matrix and the model generated by the LeNet-5 network structure exceed the Arduino's memory limit. Therefore, we propose an improved neural network structure of LeNet-5, which changes the number of convolution kernels and the number of full connection layers. The comparison between the improved network structure and the typical LeNet-5 structure is shown in Fig. 3.

4.1 Accuracy evaluation

MoTransFrame is compared with other compressed machine learning models, including Bonsai [Kumar, Goyal and Varma (2017)], neural network pruning [Chowdhery, Warden, Shlens et al. (2019)], decision jungles [Shotton, Sharp, Kohli et al. (2013)] and sparse w/o pruning [Fedorov, Adams, Mattina et al. (2019)], as shown in Fig. 4. None of the above algorithms had done a detailed analysis of the energy consumption of model reasoning.



(a) (b)

Figure 4: Four-stage pipeline for Model Transfer Framework

To solve the problem of transplanting deep learning model to edge computing platform, on the one hand, it is necessary to reduce the loss of model precision, on the other hand, it is also the most important to reduce the energy cost of running the model in edge computing equipment. Among the above efficient compression models, the MoTransFrame can transplant model to the Arduino mega board with higher accuracy in the same number of parameters.

4.2 Energy assumption evaluation

Fig. 5 shows the structure of a general energy model for evaluating energy consumption [Moons, Goetschalckx, Van Berckelaer et al. (2017)] of typical CNNs reasoning. The global energy inferred each time is the sum of the energy consumed by communication with flash and SRAM and the energy of the inference calculation itself.

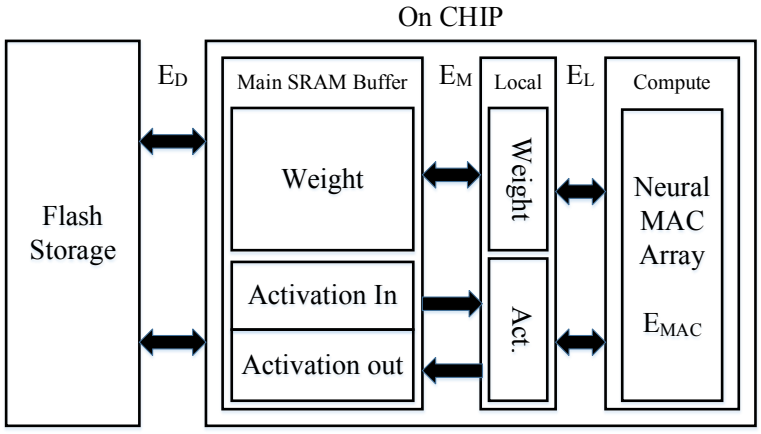


Figure 5: High-level overview of the system architecture

The total energy consumed by each network reasoning is:

$$E_{\inf} = E_{DRAM} + E_{HW} \tag{2}$$

Using the model described in Zhang et al. [Zhang, Davoodi and Hu (2018)], the small local SRAM or register file buffer contains the currently used weights and activations. We compute the inference energy by the following equation:

$$E = E_{MAC} + \mathrm{E}_{\mathrm{DRAM}} + E_{SRAM} \tag{3}$$

$$\begin{cases} E_{MAC} = E(M) \times M \\ E_{\mathrm{DRAM}} = E(D) \times (P + \mathrm{Im}\, ageSize) \\ E_{\mathrm{SRAM}} = E(S) \times P + 2E(S) \times A \end{cases} \tag{4}$$

In which E_{MAC} represents the energy consumed by the MAC array, $E(M)$ is the energy consumed by one MAC operation, and M is MAC operations in the CNN model. E_{DRAM} represents the energy consumed by DRAM. $E(D)$ is energy for fetching one word of a fixed quantization from DRAM, and P represents the total number of parameters (same as distinct edge weights) in the considered CNN. $ImageSize$ is the number of words used to represent the image. The term E_{SRAM} is the energy consumed by the SRAM realization of the activation and weight buffers. $E(S)$ is energy for one SRAM access. P is the number of parameters, and A is the number of activations in the considered CNN.

In our experiments, we used the numbers based on Horowitz [Horowitz (2014)] reported, summarized in Tab. 2 and Tab. 3. Reference data of these two tables are calculated based on 45 nm chips. We assumed 32-bit floating point MAC unit is used to perform both multiplication and addition. (So, each multiplication was followed by addition as is the case for matrix multiplication in each layer.)

Table 2: Energy consumption of multiplication and accumulations

Operation	MUL	ADD
8-bit Integer	0.2pJ	0.03pJ
16-bit Integer	3.1pJ	0.1pJ
16-bit Floating Point	1.1pJ	0.4pJ
32-bit Floating Point	3.7pJ	0.9pJ

Table 3: Energy consumption of memory access

Memory size	64-bit Cache
8 K	10 pJ
32 K	20 pJ
1 M	100 pJ
DRAM	1.3-2.6 nJ

In Fig. 6, with the limitation of on-chip memory capacity, when the cost of the DRAM access becomes a major part of total energy consumption, MoTransFrame is the minimum energy solution for all precision targets. In high-efficiency machine learning algorithms, both MoTransFrame and Bonsai have the best energy efficiency, but MoTransFrame we proposed has the highest energy efficiency in the deep learning model inference.

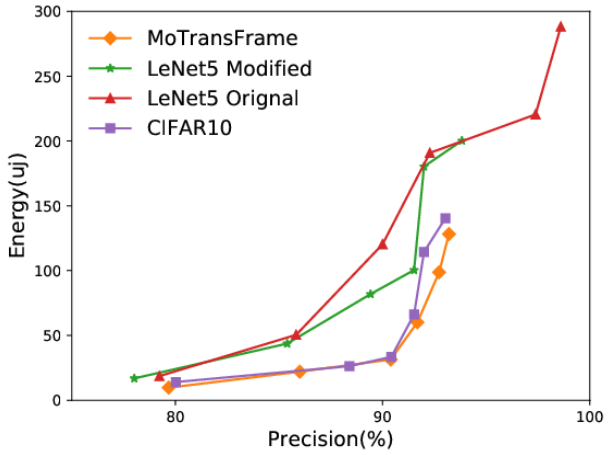


Figure 6: Comparison of energy consumption with different CNN architectures

5 Conclusions

Edge nodes are the most widely deployed computing platform, but it is almost impossible to implement large and complex CNN model with a small memory development board. The main reason is the irreconcilable contradiction between the complexity of the deep learning model and the scarcity of node resources. We describe the reasons why the low-resource edge node is positioned as a deep learning deployment host. We have proven that, contrary to previous statements, it is entirely possible to design a CNN for an MCU with a minimum of 8 KB of RAM. Besides, the perspective of accuracy, MoTransFrame has higher accuracy than other algorithms under the same number of parameters. The energy consumption for reasoning is also lower than other algorithms.

As a result, the research conducted demonstrates that it is completely feasible to run the most advanced deep learning algorithms on edge node devices for all purposes. We have proposed different platform transplant frameworks based on model inference rules. Not only can deep learning models be transplanted efficiently, but model accuracy can also be

preserved to the greatest extent. The capabilities of artificial intelligence are extending to the edge. Billions of things will connect with each other, and intelligent edge computing will bring inestimable value. We hope that MoTransFrame can inspire readers to think and explore more on edge computing and propose systematic and universal solutions.

Funding Statement: This work is supported by The National Key Research and Development Program of China (2018YFB1800202, 2016YFB1000302, SQ2019ZD090149, 2018YFB0204301), the CETC Joint Advanced Research Foundation (6141B08080101), The Major Special Science and Technology Project of Hainan Province (ZDKJ2019008), The New Generation of Artificial Intelligence Special Action Project (AI20191125008).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Cai, Z. W.; He, X. D.; Sun, J.; Vasconcelos, N. (2017): Deep learning with low precision by half-wave Gaussian quantization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5918-5926.

Chen, H. W.; Yu, J. P.; Liu, F.; Cai, Z. P.; Xia, J. et al. (2019): Archipelago: a medical distributed storage system for interconnected health. *IEEE Internet Computing*.

Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. (2017): A survey of model compression and acceleration for deep neural networks. arXiv:1710.09282.

Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.; Rhodes, R. (2019): Visual wake words dataset. arXiv:1906. 05721.

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. (2016): Binarized neural networks: training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv:1602.02830.

Fedorov, I.; Adams, R. P.; Mattina, M.; Whatmough, P. (2019): Sparse: Sparse architecture search for CNN's on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, pp. 4978-4990.

Guo, Y. T.; Liu, F.; Xiao, N.; Chen, Z. G. (2019): Task-based resource allocation bid in edge computing micro datacenter. *Computers, Materials & Continua*, vol. 61, no. 2, pp. 777-792.

Han, S.; Kang, J. L.; Mao, H. Z.; Hu, Y. M.; Li, X. et al. (2017): ESE: Efficient speech recognition engine with sparse LSTM on FPGA. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75-84.

Han, S.; Mao, H. Z.; Dally, W. J. (2015): Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv:1510. 00149.

Han, S.; Pool, J.; Tran, J.; Dally, W. J. (2015): Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, pp. 1135-1143.

Horowitz, M. (2014). 1.1 computing's energy problem (and what we can do about it).

IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp. 10-14.

Huang, M. F.; Zhang, K.; Zeng, Z. W.; Wang, T.; Liu, Y. X. (2020): An AUV-assisted Data Gathering Scheme based on Clustering and Matrix Completion for Smart Ocean. *IEEE Internet of Things Journal*.

Kumar, A.; Goyal, S.; Varma, M. (2017): Resource-efficient machine learning in 2 KB RAM for the internet of things. *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1935-1944.

Li, D. W.; Wang, X. L.; Kong, D. (2018): Deeprebirth: accelerating deep neural network execution on mobile devices. *Thirty-Second AAAI Conference on Artificial Intelligence*.

Li, E.; Zhou, Z.; Chen, X. (2018): Edge intelligence: on-demand deep learning model co-inference with device-edge synergy. *Proceedings of the Workshop on Mobile Edge Communications*, pp. 31-36.

Li, H.; Ota, K.; Dong, M. X. (2018): Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network*, vol. 32, no. 1, pp. 96-101.

Liu, C.; Cao, Y.; Luo, Y.; Chen, G.; Vokkarane, V. et al. (2017): A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 249-261.

Liu, F.; Tang, G. M.; Li, Y. H. Z.; Cai, Z. P.; Zhang, X. Z. et al. (2019): A Survey on edge computing systems and tools. *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537-1562.

Liu, S. H.; Cai, Z. P.; Xu, H.; Xu. M. (2015): Towards security-aware virtual network embedding. *Computer Networks*, vol. 91, pp. 151-163.

Moons, B.; Goetschalckx, K.; Van Berckelaer, N.; Verhelst, M. (2017): Minimum energy quantized neural networks. *51st Asilomar Conference on Signals, Systems, and Computers*, pp. 1921-1925.

Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. (2016): Xnor-net: imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision*, pp. 525-542.

Shotton, J.; Sharp, T.; Kohli, P.; Nowozin, S.; Winn, J. et al. (2013): Decision jungles: compact and rich models for classification. *Advances in Neural Information Processing Systems*, pp. 234-242.

Tang, Q.; Wang, K. Z.; Song, Y.; Li, F.; Park, J. H. (2019): Waiting time minimized charging and discharging strategy based on mobile edge computing supported by software defined network. *IEEE Internet of Things Journal*.

Teng, H. J.; Ota, K.; Liu, A. F.; Wang, T.; Zhang, S. B. (2020): Vehicles joint UAVs to acquire and analyze data for topology discovery in large-scale IoT systems. *Peer-to-Peer Networking and Applications*, 1-24.

Tseng, V. W.; Bhattachara, S.; Fernández-Marqués, J.; Alizadeh, M.; Tong, C. et al. (2018): Deterministic binary filters for convolutional neural networks. *International Joint Conferences on Artificial Intelligence Organization*.

Wu, H. J.; Zhang, J.; Cai, Z. P.; Liu, F.; Li, Y. Y et al. (2020): Towards energy-aware caching for intelligent connected vehicles. *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1-10.

Yazici, M. T.; Basurra, S.; Gaber, M. M. (2018): Edge machine learning: enabling smart internet of things applications. *Big data and Cognitive Computing*, vol. 2, no. 3.

Yu, G.; Cai, Z. P.; Wang, S. Q.; Chen, H. W.; Liu, F. et al. (2019): Unsupervised online anomaly detection with parameter adaptation for KPI abrupt changes. *IEEE Transactions on Network and Service Management*.

Zhang, B.; Davoodi, A.; Hu, Y. H. (2018): Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 836-848.

Zhang, J, M.; Wang, W.; Lu, C. Q.; Wang, J.; Sangaiah, A. K. (2019): Lightweight deep network for traffic sign classification. *Annals of Telecommunications*, pp. 1-11.

Zhang, J. M.; Lu, C. Q.; Li, X. D.; Kim, H. J.; Wang, J. (2019): A full convolutional network based on DenseNet for remote sensing scene classification. *Mathematical Biosciences and Engineering*, vol. 16, no. 5, pp. 3345-3367.

Zhang, J.; Wang, X. L.; Li, D. W.; Wang, Y. L. (2018): Dynamically hierarchy revolution: Dirnet for compressing recurrent neural network on mobile devices. arXiv:1806.01248.