AIoT Solution Survey and Comparison in Machine Learning on Low-cost Microcontroller

Hoang-The Pham[1], Minh-Anh Nguyen[1], Chi-Chia Sun[12]
[1]Department of Electrical Engineering
[2]Smart Machinery and Intelligent Manufacturing Research Center
National Formosa University, Yunlin, Taiwan

Abstract— **Neural Networks, especially Convolutional Neural Network [1] are becoming increasingly popular in IoT edge devices today for executing data analytics right at the source without transmitting to Cloud Computing centers. So that it will be reduced latency as well as energy consumption for data communication. In this paper, we will compare CMSIS-NN and uTensor: low energy consumption microcontrollers. Most classification tasks have always-on, and real-time requirements, which limits the total number of operations per neural network inference. So that, with image classification model, microcontrollers will execute lower frame per second than GPU and embedded CPU. CMSIS-NN is a collection of efficient kernels which was developed to maximize the performance and minimize the memory footprint of Neural Network applications. Allow deploy machine learning models on ARM Cortex-M processors for intelligent IoT edge devices.**

I. INTRODUCTION

Connected devices or Internet of Things (IoT) have been rapidly increasing over the past few years and are predicted to reach 1 trillion across various market segments by 2035. As we known, the number of the IoT devices more and more increases, this will place a considerable burden on the network bandwidth, so that latency will be challenging when running the AIoT applications [2] in the future. Dependency on the cloud computing makes it harder to deploy AIoT application in areas with low and unreliable network connectivity. The solution for this problem is edge computing, data will not be transmitted to cloud, they will be processed and executed right at the source after collected by sensors of IoT devices. Thus, this solution will help us to reduce the latency as well as saving energy for data communication.

Deep Neural Network (DNN) [3] based solutions have performed very high accuracies for many complex applications such as computer vision, natural language processing, image classification, optical character recognition, object detection, and speech recognition, etc. Due to complex computation and hardware resource requirements, these executions of Neural Networks must be performed on cloud computing which has high performance server CPUs as well as GPUs. As mentioned above, it will add latency to the AIoT applications. Executing right at the source of data on small microcontrollers can reduce the overall latency and energy consumption of data communication between the IoT devices and the cloud. However, we have to deal with these challenges when deploy Neural Network model on microcontrollers in the edge side.

In this paper, we have surveyed CMSIS-NN [4][5], which is a collection of efficient neural network kernels developed to maximize the performance and minimize the memory footprint of neural networks on ARM Cortex-M7 processor. Furthermore, we will present the workflow to deploy neural networks on Cortex-M7 with and without CMSIS-NN. Caffe [6] is the deep learning framework which is made with expression, speed, and modularity in mind by Berkeley AI Research. We used Caffe for training the image classification models. The models trained will be quantized to reduce the size of the neural network and avoid floating point computations, that are more computationally expensive. Then, we used tools to convert the model weight to C++ code that can be compiled and run on the microcontroller. For the Cortex-M7's hardware, STM32F746ZGT6U [7] has been selected for comparing for performing image classification, especially neural networks model trained on CIFAR-10, MINIST, SVHN datasets.

II. CONTROL METHODS

II.1. Model with CMSIS-NN

CMSIS-NN is a collection of optimized neural network functions for ARM Cortex-M core microcontrollers enabling neural networks and machine learning being pushed into the end node of AIoT applications. It has implemented popular neural network layer types, such as convolution, depth separable, fully connected, pooling and activation (ReLU). Supporting a model trained with a popular framework such as TensorFlow, Caffe. The weights and biases will first be quantized to 8-bit or 16-bit integers then deployed to the microcontroller. The best performance was achieved by leveraging SIMD instructions features of the CPU to improve parallelism available for Cortex-M4 and Cortex-M7 core microcontrollers.

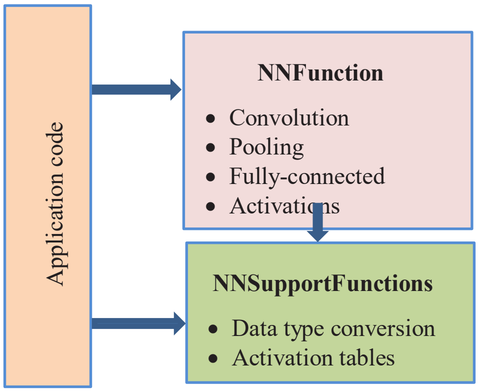

Figure. 1. Block Diagram CMSIS-NN

From pre-trained model CIFAR-10 by Caffe, the model will be translated to C++ file to be able to deploy on ARM. Figure 1 show how to build model with CMSIS-NN and deploy it on MCU is follow:

1) Quantize the model: Once we have the CIFAR-10 trained model, we need to optimize it for microcontroller. We use ARM quantization script to convert the model weights and 14 activations from 32-bit floating point to an 8-bit and fixed-point format. This work will reduce the size of the network and avoid floating point computations.
2) Convert the model: Then we need convert the model into C++ file that we can include it into image recognition application. Use generate script to get the quantization parameters and network graph connectivity and generates the code consisting of NN function calls.
3) Build image recognition application: We need include file to main

978-1-7281-3038-5/19/$31.00 ©2019 IEEE

file and add classification capabilities to use neural network on ARM. Modify function and call run command to run neural network. We still need image as input of NN so we resize and translate picture to array can work in C++.

4) Deploy on ARM-Cortex M: Use MbedOS [8] compiler to generate binary file and upload to the MCU processor.

II.2. Model with MicroTensor

MicroTensor (uTensor) [9] ARM’s early entrant into edge machine learning, takes TensorFlow models and compiles into highly efficient code for edge processing. uTensor converts machine learning models to readable and self-contained C++ source files, to simplify the integration with any embedded project. It is especially designed for low-power, constrained embedded devices, and it has deep roots in TensorFlow and MbedOS. From this merger, we have a great opportunity to bring uTensor’s innovations to TensorFlow and ensure it is easy for all developers to use and support a wide range of ARM Cortex-M hardware. All of developers share a common vision to bring machine learning to the edge. We are looking forward to creating a state-of-the-art micro-inference framework together.

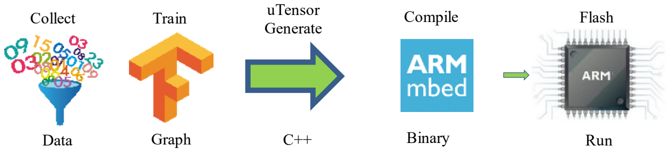

Figure. 2. Workflow of uTensor

This workflow from is Figure 2 is different from deploying neural network model with CMSIS-NN. Instead of training by Caffe, this model will be trained by TensorFlow [10]. The neural network models after trained will be convert to C++ code by tools called uTensor as mentioned above. Here is workflow for deploying Neural Network model CIFAR-10 to ARM Cortex-M7.

1) Training Neural Network model by TensorFlow： From CIFAR-10 dataset, we use TensorFlow and images from CIFAR-10 dataset to train the network. Then convert trained model to model file to use for uTensor-cli.
2) Translate the Neural Network model to C++ code by uTensor-Cli: Use uTensor-cli to identify the output nodes and generate the C++ files from CIFAR-10 model.
3) Compile and flash project to ARM Cortex-M by Mbed OS compiler: We need include file to main file and add classification capabilities to use neural network on ARM. Modify this function and call run command to run neural network. We still need image as input of NN so we resize and translate picture to array can work in C++.
4) Deploy on ARM Cortex-M: Use MbedOS compiler to generate binary file and burn to the board.

III. EXPERIMENTAL RESULTS

Next, we applied these two different ways to deploy neural network models with three datasets. Three common datasets that we used are MNIST (handwritten digits), CIFAR-10, Street View House Number (SVHN). Two models have not yet completed are SVHN without CMSIS-NN and MNIST with CMSIS-NN. The model MNIST with CMSIS-NN was not completed because of it has two Fully Connected layers which generated a lot of weights could not fit the microcontroller’s memory. We could only complete measuring the accuracy and the performance of microcontroller when running neural network model. Due to microcontroller’s memory, we could not put in its memory many input images. Accuracy was measured by randomly run 100 samples and calculate the error rates. The performance is measured by calculate how long does it took the model to predict label. Before running the model, we initiated the time object then started counting the timer, let the model run, then stopped the timer and calculate how many seconds to complete the task. We calculated the performance by frames per second (FPS). From table 1, we can see model with uTensor gave us results of performance is better than model with Caffe & CMSIS-NN.

Table 1. Comparison between Caffe & CMSIS-NN and uTensor

	Caffe & CMSIS-NN		uTensor	
Model/ Datasets	Error Rate (%)	Performance	Error Rate (%)	Performance
CIFAR-10	26%	9.09 FPS	15%	2.22 FPS
MNIST	Out of memory	Out of memory	10%	7.69 FPS
SVHN	32%	8.33 FPS	Out of memory	Out of memory

IV. CONCLUSION

In this paper, we were running the neural network with pre-defined input data which is no reality when considering variety choices of sensors, camera, microphone, accelerometer all can be easily integrated with the microcontroller to acquire real-time data form the environment. There are endless possibilities when this neural network framework is leveraged to process those data and extract useful information. The Internet of Things is slowly permeating every aspect of our lives. By using CMSIS-NN, we can easily integrate Machine Learning on ARM then connect to AIoT system to make an intelligent AIoT system. There is an increasing interest in deploying the deep learning algorithms on low-power edge devices such as ARM Cortex-M microcontroller systems. CMSIS-NN can raise speed, and reduce power cost of system so it can help us easily use in many cases.

REFERENCES

[1] Keiron O’Shealand Ryan Nash, “An Introduction to Convolutional Neural Networks,” arXiv:1511.08458, 2015.

[2] Marjan Gusev, Schahram Dustdar, “Going Back to the Roots—The Evolution of Edge Computing, An IoT Perspective,” IEEE Internet Computing, Vol. 22, No. 2, pp. 1-5, 2018.

[3] Michael A. Nielsen, “Neural Networks and Deep Learning,” Determination Press, pp. 167-176, 2015.

[4] Liangzhen Lai, Naveen Suda, and Vikas Chandra, “CMSIS-NN: Efficient Neural Network Kernels for ARM Cortex-M CPUs,” arXiv:1801.06601, 2018.

[5] Liangzhen Lai and Naveen Suda, “Enabling Deep Learning at the IoT Edge. International Conference on Computer-Aided Design,” pp. 135:1-135:6, 2018.

[6] Delia Velasco-Montero, Jorge Femández-Bemi, Ricardo Carmona-Gálán, Angel Rodríguez-Vázquez, “On the Correlation of CNN Performance and Hardware Metrics for Visual Inference on a Low-Cost CPU-based Platform,” IEEE International Conference on Systems, Signals and Image Processing, pp. 250-252, 2019

[7] “STM32F746ZG datasheet” by STMicroelectronics company https://www.st.com/resource/en/datasheet/stm32f746zg.pdf

[8] “Mbed OS” by Arm Limited available here https://os.mbed.com/

[9] Neil Tan, “uTensor- AI inference library based on Mbed and TensorFlow,” https://github.com/uTensor/uTensor

[10] Tom Hope, Yehezkel S. Resheff and Itay Lieder, “Learning TensorFlow,” pp. 1-21 O’Reilly Media, 2017.

978-1-7281-3038-5/19/$31.00 ©2019 IEEE