# Image Classification on Resource-Constrained Microcontrollers

1st Seungtae Hong
*Intelligent Device & Simulation Research Section*
*Electronics and Telecommunications Research Institute*
*and*
*University of Science and Technology*
Daejeon, Korea
sthong@etri.re.kr

2nd Gunju PARK
*Intelligent Device & Simulation Research Section*
*Electronics and Telecommunications Research Institute*
Daejeon, Korea
parkgj@etri.re.kr

3rd Jeong-Si Kim
*Intelligent Device & Simulation Research Section*
*Electronics and Telecommunications Research Institute*
Daejeon, Korea
sikim00@etri.re.kr

***Abstract*—Recently, as IoT devices have become popular, research to perform deep learning in small devices such as microcontrollers has been attempted. Microcontrollers have very limited resources compared to edge devices such as mobiles. Therefore, in order to perform deep learning-based image classification in a microcontroller, an optimization technique considering HW constraints is required. To this end, in this paper, we present a method for light weighting a model so that it can be executed in a microcontroller, and a process for distributing the lightweight model to a microcontroller. Finally, it was confirmed that image classification can be performed in an actual microcontroller through STM32F746G-Discovery.**

***Keywords—Microcontrollers, Deep Learning, Image Classification***

## I. Introduction

With the progress of IT technology, deep learning technology is being utilized in a multitude of domains [1][2]. In particular, with the recent widespread use of IoT devices, research to perform deep learning in micro-devices such as microcontrollers is being attempted [3].

A microcontroller consists of a CPU, memory, and input/output features on a single chip. In addition, microcontrollers have very few available resources, such as memory (SRAM) within a few hundred KB and flash within several MB. However, since microcontrollers are very cheap and consume very little power, they can be used in many real-life situations.

To this end, this paper proposes an optimization technique for image classification on resource-limited microcontrollers. The proposed optimization technique presents a model conversion technique for deploying a pretrained model in the server to a microcontroller. In addition, the proposed optimization method presents a preprocessing method for optimizing inference speed in microcontrollers. To verify the optimization method proposed in this paper, image classification is performed on the Cifar-10 dataset using TensorFlow Lite for Microcontrollers on the STM32F746G-Discovery board [4].

## II. Related Works

TensorFlow Lite for Microcontrollers (=TFLM) is a subcomponent of TensorFlow Lite for performing machine learning on resource-constrained microcontrollers. TFLM takes as input a model transformed through TensorFlow Lite. At this time, TensorFlow Lite converts the pre-trained model to FlatBuffer [5] format. FlatBuffer is a cross-platform serialization interface proposed by Google. FlatBuffer can be accessed directly without separate parsing or unpacking, and has the advantage of being usable on various platforms without any dependencies.

Since the microcontroller does not have a file system, the model converted to FlatBuffer is included in the source code in the form of a C array. The model is then compiled along with other source code, built in binary form and stored in the flash on the microcontrollers.

TFLM runs inference on the microcontroller through an interpreter API in the form of a C/C++ language. For initialization and control of the microcontroller, the API of the BSP (Board Support Package) should be used separately from the interpreter API of TFLM.

## III. Image Classification On Resource-Constrained Microcontrollers

The procedure of the optimization technique for image classification on a microcontroller is shown in Figure 1.

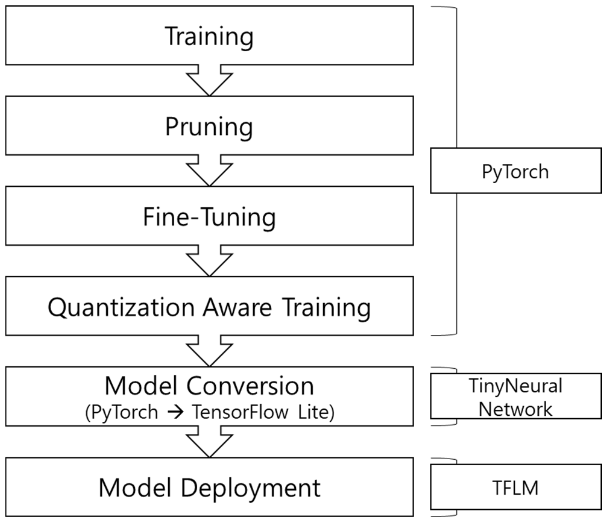


Fig. 1. The procedures of optimization techniques for image classification on microcontrollers

TFLM natively takes as input models from TensorFlow Lite that have been converted from TensorFlow. However, various model lighting techniques such as pruning and quantization are currently being released based on PyTorch. Therefore, the proposed technique proceeds with PyTorch-based training, pruning, and quantization aware training. The proposed method uses TinyNeuralNetwork [6] to convert a PyTorch model that has completed quantization aware training into a TensorFlow Lite model. Finally, TensorFlow Lite models are converted to C array using the *xxd* command. The *xxd* command is a utility available on Linux and Unix-like operating systems used to create and manipulate hexadecimal and binary representations of files.

We used MobileNet v2 [7], which was redesigned to fit the Cifar-10 dataset, and modified the output channels and expansion ratio considering the specifications of the microcontroller. The modified MobileNet v2 has an input size of 32 * 32 * 3 (width * height * channel) and consists of 16 bottlenecks.

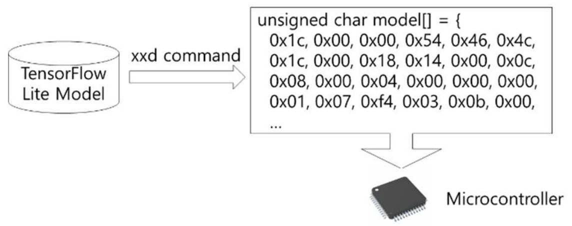
After training is finished, One Shot Channel Pruning [8] is performed to reduce the parameters of the model. One Shot Channel Pruning has the advantage of shorter execution time compared to existing pruning techniques. After pruning, fine-tuning is performed to improve the accuracy of the model.

In general, quantization techniques are divided into post-training quantization that can be performed without retraining and quantization-aware training that performs quantization while performing training. In this paper, we use quantization-aware training to mitigate the loss of accuracy due to model pruning.

To deploy the quantized model to the microcontroller, we need to convert the model with TensorFlow Lite. To this end, in this paper, we transform the trained model using Alibaba's TinyNeuralNetwork. Typically, ONNX (Open Neural Network Exchange) is used when converting models trained with PyTorch to TensorFlow Lite. However, while the model conversion process using ONNX is difficult, the model conversion process using TinyNeuralNetwork is relatively easy.

When inference is performed in a microcontroller, the pre-processing of the input image must be minimized to speed up the inference. To this end, when converting a PyTorch model to a TensorFlow Lite model, the input and output of the neural network use quantized input (=signed int8). That is, quantization is applied to all inputs and outputs as well as the hidden layer of the neural network. Through this, each pixel value of the image obtained from the camera module on the microcontroller can be directly used as a quantization input.

To deploy a TensorFlow Lite model to a microcontroller, it must be converted into an array with hexadecimal values using the *xxd* command. Figure 2 shows an example of converting a TensorFlow Lite model to a hexadecimal array through the *xxd* command.



```
unsigned char model[] = {
  0x1c, 0x00, 0x00, 0x54, 0x46, 0x4c,
  0x1c, 0x00, 0x18, 0x14, 0x00, 0x0c,
  0x08, 0x00, 0x04, 0x00, 0x00, 0x00,
  0x01, 0x07, 0xf4, 0x03, 0x0b, 0x00,
  ...
```

Fig. 2. An example of converting a hexadecimal array of TensorFlow Lite models

Apart from TFLM, BSP (Board Support Package) must be used to initialize and control the microcontroller. BSP provides user APIs that can directly control HW such as CPU clock setting and sensor initialization. In addition, in the microcontroller, the inference speed can be optimized using the HW control API of the BSP. For example, STM32 provides cache control APIs (e.g., SCB_EnableICache(), SCB_EnableDCache()), through which microcontrollers can improve inference speed.

## IV. Experiments

### A. Experimental Setup

To prove the validity of the proposed technique, we implemented an image classifier on a representative development board, STM32F746G-Discovery. Table 1 shows the HW specifications of STM32F746G-Discovery. To build the source code for the microcontroller, we used STM32CubeIDE (ver. 1.11.0) officially provided by STM. STM32CudeIDE has a built-in C/C++ compiler. We performed performance evaluation by specifying the -Ofast option to optimize inference speed.

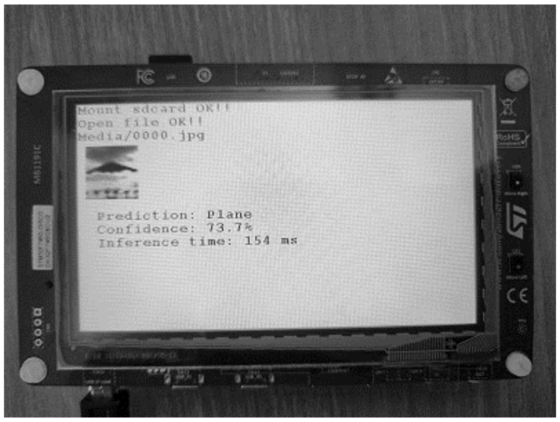
TABLE I. HW specifications of STM32F746G-Discovery

| Type | Specification |
|---|---|
| CPU | STM32F746NG (ARM Cortex-M7)<br>- Single Core (216 MHz) |
| SRAM | 320 KB (User SRAM:256KB) |
| Flash memory | 1 MB |
| Power | 3.3 V or 5 V |
| Supported devices | 4.3” LCD display<br>Micro SD card<br>On-board ST-LINK/V2-1 debugger/programmer |

### B. Evaluation Results

STM32F746G-Discovery supports LCD screen and external micro SD card. In this performance evaluation, the Cifar-10 test dataset is saved on a micro SD card, and inference is performed by loading the saved image data. Then, the inference result is output on the LCD screen.

Figure 3 shows the screen of running the image classifier on the STM32F746G-Discovery. 10 images randomly extracted from the Cifar-10 test dataset are stored in the micro SD card. Inference is performed by sequentially loading images by pressing user buttons below the LCD screen. Figure 3 shows the screen of running the image classifier on the STM32F746G-Discovery.



```
Mount sdcard OK!!
Open file OK!!
Media/0000.jpg

Prediction: Plane
Confidence: 73.7%
Inference time: 154 ms
```

Fig. 3. Screen for performing inference on STM32F746G-Discovery

Table 2 shows the change trend of FLOPs (FLoating point Operations Per Second) and validation accuracy in the model conversion process We trained for 300 epochs on the server and got top-1 accuracy of 92.26%. After training, pruning and fine-tuning were performed to reduce the weight of the model, and FLOPs were reduced by 40.04%. On the other hand, the accuracy after pruning decreased to 87.16%. Therefore, in order to compensate for the accuracy lost through pruning, quantization is performed through quantization-aware training. When quantization was performed through quantization-aware training, it was confirmed that the accuracy improved by 1.33% compared to the model with pruning and fine-tuning.

TABLE II. TRENDS IN FLOPS AND ACCURACY BY MODEL CONVERSION PROCESS

| **Step** | **FLOPs** | **Validation Accuracy (Top-1)** |
|---|---|---|
| Traing | 15,448,512 | 92.26 % |
| Pruning + Fine-tuning | 9,262,416 (-40.04 %) | 87.16 % (-5.1%) |
| Quantization-aware training | 9,262,416 (-40.04 %) | 88.49 % (-3.77 %) |

Table 3 shows the size difference between SRAM and Flash depending on whether TFLM is included or not. Without TFML means that the binary running on the microcontroller contains only basic BSP and libraries for loading image files (e.g., FatFs, LibJPEG). If TFLM is included in binary, it also includes a model for inference.

TABLE III. LIST OF RECOGNIZERS FOR PERFORMANCE EVALUATION

| **Type** | **SRAM (Max: 320 KB)** | **Flash (Max: 1 MB)** |
|---|---|---|
| Without TFLM (including FatFs, LibJPEG) | 4.18 KB (1.3%) | 88.89 KB (8.68%) |
| With TFLM (including model) | 144.94 KB (45.29%) | 685.28 KB (66.92%) |

In order to perform inference using TFLM, activation memory for each operation is required. The activation memory is allocated to the SRAM of the microcontroller. In contrast, the model converted to hexadecimal is allocated to flash memory. On the other hand, TFLM uses an interpreter method to perform inference on various microcontrollers. Therefore, TFLM requires additional memory for the interpreter method along with activation memory.

## V. CONCLUSION

In this paper, we proposed an optimization technique for image classification using deep learning in a microcontroller. Microcontrollers have very limited resources compared to portable edge devices such as mobiles. Therefore, in order to perform deep learning-based image classification in a microcontroller, resource limitations of the microcontroller must be considered.

In this paper, we presented the model conversion process to deploy the trained model to the microcontroller. In addition, it was confirmed that image classification can be performed through the proposed optimization technique by performing inference in STM32F746G-Discovery.

We plan to conduct research on performing image classification on large-sized input data such as the ImageNet dataset.

## ACKNOWLEDGMENT

This research was supported by the Challengeable Future Defense Technology Research and Development Program through the Agency For Defense Development(ADD) funded by the Defense Acquisition Program Administration(DAPA) in 2022(No.915062201)

## REFERENCES

[1] H. H. Bu, N. C. Kim, and S. H. Kim, "Content-based image retrieval using a fusion of global and local features," ETRI Journal, vol. 45, no.3, 2023

[2] S. Seo, and H. Jung, "A robust collision prediction and detection method based on neural network for autonomous delivery robots," ETRI Journal, vol. 45, no. 2, 2023.

[3] J. Lin, W. M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," Advances in Neural Information Processing Systems (NIPS), 2020

[4] STMicroelectronics, Discovery kit with STM32F746NG MCU, https://www.st.com/en/evaluation-tools/32f746gdiscovery.html

[5] Google, FlatBuffers, https://github.com/google/flatbuffers

[6] alibaba, TinyNeuralNetwork, https://github.com/alibaba/TinyNeural Network

[7] chenhang98, mobileNet-v2_cifar10, https://github.com/chenhang98/ mobileNet-v2_cifar10

[8] F. Yu, C. Han, P. Wang, X. Huang, and L. Cui, "Gate trimming: One-shot channel pruning for efficient convolutional neural networks," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021.