Real-Time Image Classification and Transformation Apps on iOS by "Chainer2MPSNNGraph"

Yuki Izumi Daichi Horita Ryosuke Tanno Keiji Yanai
Department of Informatics,
The University of Electro-Communications, Tokyo
1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585 JAPAN
{izumi-y,horita-d,tanno-t,yanai}@mm.inf.uec.ac.jp

Abstract

We propose a DNN code generator, "Chainer2MPSNNGraph", which converts DNN models trained by Chainer [1] into Swift codes utilizing MPSNNGraph APIs. Note that MPSNNGraph API is a part of the iOS Metal Performance Shader library which is a library to utilize a GPU inside an iPhone. That is, the proposed tool enables us to develop GPU-powered DNN applications running on iPhone easily. In addition to introduction of "Chainer2MPSNNGraph", we also show some real-time image recognition and transformation applications implemented with "Chainer2MPSNNGraph": image classification with AlexNet and VGG16, multi-style transfer and food translation.

1 Introduction

DNN (Deep Neural Network) have made AI applications advance greatly. For real-time or interactive mobile applications, on-device execution of DNN is really needed. To do that, some recent smartphones such as iPhone X have GPUs for deployment of DNN. In case of iOS, Apple provides an API library for GPU-powered computation on iPhone, Metal Performance Shader (MPS). The MPS in iOS 11 or later contains Neural Network APIs which is called "MPSNNGraph" APIs. They enables has GPU-powered forward computation of basic neural network layers such as a fully connected (FC) layer, a convolutional (CONV) layer, batch normalization and ReLU activation function.

To utilize a GPU inside iPhone, CoreML API and Core ML tools are officially provided by Apple. CoreML API is a high level library for machine learning which is built upon both low-level GPU libraries, MPS, and low-level CPU-based computation library, Accelerate. The Core ML tools translate DNN models trained by Keras or Caffe into CoreML format models, and the CoreML format models can be deployed using CoreML API. Although CoreML can execute trained models on iPhone GPU easily, it is black-boxed and not open-sourced. For research purpose, this characteristics is unwelcome for exploring efficient on-device deployments of DNN models.

Then, in this paper, we propose a DNN code generator, "Chainer2MPSNNGraph", which converts DNN models trained by Chainer into Swift codes utilizing MPSNNGraph APIs. The generated code can be modified and optimized by hand, which is totally different from black-boxed CoreML. In addition, DNN models are represented in Swift codes after code generation, and the generated codes are compiled by the Swift compiler into the executable codes on iPhone. Therefore, it has possibility to run DNN models more efficiently than CoreML. We examine this point with image classification networks in the experiments. In addition, we show some real-time image recognition and transformation applications implemented with "Chainer2MPSNNGraph": image classification with AlexNet and VGG16, multi-style transfer and food translation.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

2 Related Work

Some DNN frameworks provide extension libraries for smartphone deployment such as Caffe iOS/Android, Caffe2, PyTorch and Tensorflow Lite. Except Tensorflow Lite, all the frameworks employ the library for fast computation of DNN networks, NNPACK [1]. NNPACK supports multi-threading and ARM SIMD (Neon) instructions for efficient computation of generic matrix multiplication (GEMM).

Tensorflow Lite uses the Android Neural Network (ANN) API which is available at Android OS 8.1 or later. Although the default implementation of ANNs is CPU-based, some Android phone vendor implements ANN as being GPU-based. For example, Qualcom Snapdragon 845 officially supports ANN.

As some other works on on-device DNN deployment, "Chainer2C with Accelerate" [2] was proposed. The authors proposed a C code generator of DNN models trained with Chainer, and achieved 26ms at fastest for one image recognition by Network-in-Network on iPhone7plus. They employed the CPU-based Accelerate library in iOS and multi-threading for efficient computation of DNNs.

In this paper, we propose a DNN code generator which converts CNN models trained with Chainer into a Swift code employing a GPU-based computation library. To do that, we follow [2] and replace C with Swift as a target language and CPU-based iOS Accelerate with GPU-based iOS MPS NNGraph as a computation library.

3 Method

"Chainer2MPSNNGraph" converts the model trained with Chainer [1] into a Swift code and a parameter file.

Chainer creates a model graph dynamically at the time of forwarding computation, so that we read the model trained by Chainer and feed-forward the model once with the Chainer module, and then the proposed library, "Chainer2MPSNNGraph", which is implemented as an extension of the Chainer module written in Python, analyzes the model graph and generates a code and a parameter file for the MPSNNGraph API. Figure 1 shows a work flow for developing of a DNN iOS App from a DNN Chainer code with "Chainer2MPSNNGraph". Note that a GUI code are needed to be prepared separately, since the proposed library generates a code for only a DNN engine.

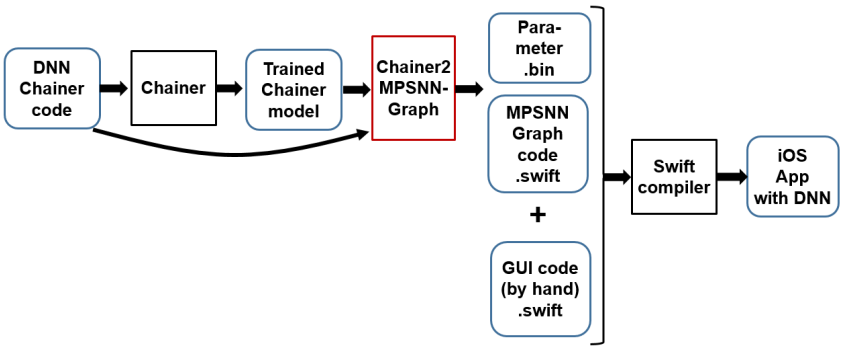

Figure 1: A work flow for developing of a DNN iOS App with "Chainer2MPSNNGraph".

4 Applications

In this section, we show some example apps developed using "Chainer2MPSNNGraph" including image classification apps and two image translation apps.

[1] https://github.com/Maratyszcza/NNPACK

```
let conv1 = MPSCNNConvolutionNode(source: conv01.resultImage,
                                  weights: DataSource("conv1", 9, 9, 3, 32, 1, useBias: true))
let relu1 = MPSCNNNeuronReLUNode(source: conv1.resultImage)
let bn1 = MPSCNNBatchNormalizationNode(source: relu1.resultImage,
                                       dataSource: DataSource2("bn1", 32))
let conv2 = MPSCNNConvolutionNode(source: bn1.resultImage,
                                  weights: DataSource("conv2", 4, 4, 32, 64, 2, useBias: true))
let relu2 = MPSCNNNeuronReLUNode(source: conv2.resultImage)
let bn2 = MPSCNNBatchNormalizationNode(source: relu2.resultImage,
                                       dataSource: DataSource2("bn2", 64))
let conv3 = MPSCNNConvolutionNode(source: bn2.resultImage,
                                  weights: DataSource("conv3", 4, 4, 64, 128, 2, useBias: true))
let relu3 = MPSCNNNeuronReLUNode(source: conv3.resultImage)
let bn3 = MPSCNNBatchNormalizationNode(source: relu3.resultImage,
                                       dataSource: DataSource2("bn3", 128))
```

Figure 2: An example of a generated Swift code which represents the DNN model.

4.1 Classification

We implemented image classification apps containing AlexNet and VGG16 as image classification engines.

First, we compared the GPU-based AlexNet app implemented with the proposed generator with the CPU-based AlexNet implemented with "Chainer2C" [2] regarding the processing time for one image. Table 1 shows the processing time at iPad Pro 9.7inch. The GPU-based app by the proposed generator was 3.7 times as fast as the CPU-based app by Chainer2C.

Next, for VGG-16, we compared the proposed one with CoreML and the MPS code [2] implemented by hand where only MPS API was used and no MPS NNGraph API was used. Table 2 shows the processing time at iPhone 8 Plus by three implementation. The proposed one achieved the best speed, 109.0ms, while CoreML and MPS were less, which shows the efficiency of MPSNNGraph API and compiling of the generated Swift code into a native code.

Table 1: Processing time on iPhone 8 Plus for one image classification by AlexNet.

code generator	time (ms)
Chainer2C (CPU)	134.9
Proposed (GPU)	**36.4**

Table 2: Processing time on iPhone 8 Plus for one image classification by VGG16. All the methods employ a GPU on iPhone.

method	time (ms)
CoreML	144.9
MPS	155.2
Proposed	**109.0**

4.2 Image Translation

As the other DNN applications than image classification, we implemented multi-style transfer network [3, 4], and food image transformation network [5].

4.2.1 Multi-Style Transfer

To implement a multi-style transfer app, we used the network proposed by Yanai et al. [3, 4] which is an extension of the Conv-5 ResBlock-Deconv network proposed by Johnson et al. [6] by adding a conditional input for mixing the style weights.

We compared the app by the proposed method with the app by Chainer2C and CoreML. As a result, the app implemented with "Chainer2MPSNNGraph" was the fasted for stylization of one 256x256 image as shown as Table 3. Figure 3 shows the screenshot of the implemented app, "DeepStyleCam".

[2] https://github.com/hollance/VGGNet-Metal

Table 3: Processing time on iPhone 8 Plus for one image stylization. (†Note that the implementation for CoreML was not multiple style transfer but single style transfer [6]. The time for CoreML is expected to increase a little in case of multiple style.)

method	time (ms)
Chainer2C (CPU)	138.6
CoreML (GPU) †	101.0
Proposed (GPU)	**96.3**

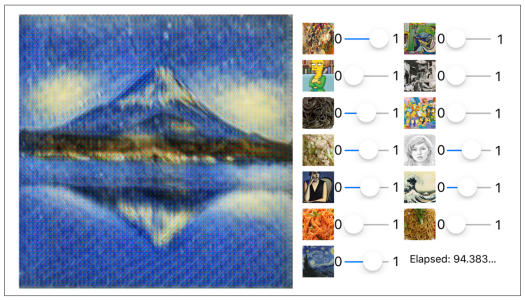

Figure 3: An screenshot of "DeepStyleCam" which is a real-time multi-style transfer app running on a iPhone GPU. A user can set the weights of the 13 pre-trained styles. Mixing multiple styles is possible.

4.2.2 Food Image Transformation

We also implemented the food image transformation network [5] was based on StarGAN [7]. The network of the transformation part is the Conv-7 ResBlock-Deconv network which is based on the network proposed by Johnson et al. [6]. Figure 4 shows examples of food image transformation which can convert a food photo into ten kinds of the pre-trained food categories. Figure 5 shows the screenshot of the implemented app, "MagicalRiceBowl". Note that one-time transformation for a 256x256 image took 115 ms, which corresponded to 9fps.

Figure 4: Examples of food image transformation.

Figure 5: An screenshot of "MagicalRiceBowl" which is a real-time food image transformation app running on a iPhone GPU.

5 Conclusions

We proposed "Chainer2MPSNNGraph" which is a code generator of MPSNNGraph API of iOS Metal Performance Shader. By using the proposed code generator, we can develop DNN apps which utilize a GPU on iPhone easily. In the experiments, we confirmed that AlexNet could runs on GPU 3.7 times faster than CPU computation, and VGG16 could run faster than CoreML. In addition, we implemented iOS apps on multi-style transfer and food image transformation utilizing a iPhone GPU using "Chainer2MPSNNGraph". We plan to demo both apps at the venue if accepted.

Note that "Chainer2MPSNNGraph" will be released at GitHub soon. In addition, we are developing "ONNX2MPSNNGraph" currently, which converts DNN models represented in the Open Neural Network Exchange (ONNX) format into Swift codes utilizing the MPSNNGraph APIs.

References

[1] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *Proc. of NIPS Workshop on Machine Learning Systems (LearningSys)*, 2015.

[2] K. Yanai, R. Tanno, and K. Okamoto. Efficient mobile implementation of a cnn-based object recognition system. In *Proc. of ACM International Conference Multimedia*, 2016.

[3] K. Yanai. Unseen style transfer based on a conditional fast style transfer network. In *Proc. of International Conference on Learning Representation Workshop Track (ICLR WS)*, 2017.

[4] K. Yanai and R. Tanno. Conditional fast style transfer network. In *Proc. of ACM International Conference on Multimedia Retrieval (ICMR)*, 2017.

[5] D. Horita, R. Tanno, W. Shimoda, and K. Yanai. Food category transfer with conditional cycle gan and a large-scale food image dataset. In *Proc. of International Workshop on Multimedia Assisted Dietary Management (MADIMA)*, 2018.

[6] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. of European Conference on Computer Vision*, 2016.

[7] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proc. of IEEE Computer Vision and Pattern Recognition*, 2018.