Caffe2C: A Framework for Easy Implementation of CNN-based Mobile Applications

Ryosuke Tanno Keiji Yanai

Department of Informatics,
The University of Electro-Communications, Tokyo
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585 JAPAN
{tanno-r,yanai}@mm.inf.uec.ac.jp

ABSTRACT

In this study, we create "Caffe2C" which converts CNN (Convolutional Neural Network) models trained with the existing CNN framework, Caffe, C-language source codes for mobile devices. Since Caffe2C generates a single C code which includes everything needed to execute the trained CNN, csCaffe2C makes it easy to run CNN-based applications on any kinds of mobile devices and embedding devices without GPUs. Moreover, Caffe2C achieves faster execution speed compared to the existing Caffe for iOS/Android and the OpenCV iOS/Android DNN class. The reasons are as follows: (1) directly converting of trained CNN models to C codes, (2) efficient use of NEON/BLAS with multi-threading, and (3) performing pre-computation as much as possible in the computation of CNNs. In addition, in this paper, we demonstrate the availability of Caffe2C by showing four kinds of CNN-base object recognition mobile applications.

CCS Concepts

•Information systems → Mobile information processing systems;

Keywords

Mobile Application, Image Recognition, Object Recognition, Deep Learning, Deep Convolutional Neural Network

1. INTRODUCTION

In recent years, Deep Neural Network (DNN) achieved remarkable progress. Its effectiveness has been demonstrated in various fields such as image recognition, audio recognition and natural language processing. Especially, in image recognition, DNN gave the best performance and outperform even humans in certain cases such as recognition of 1000 objects. In the other field than image recognition, DNN achieved the best performance in the problems on pattern recognition such as speech recognition or natural language processing. Its results confirmed the high generalization ability of DNN. Note that since in this paper we treat a mobile application of image recognition, and a Deep Convolutional Neural Network (CNN) which is a DNN consisting of convolutional layers is used in image recognition researches in general, we call a DNN in image recognition researches a CNN in this paper.

In addition, due to the rapid increase in research on CNN, many frameworks of Open-Source Software (OSS) for CNN have emerged. For example, Caffe [9], Theano [2], Tensorflow [1]. In particular, Caffe is the most popular deep learning framework due to "the fastest framework" and "providing of Python and MATLAB-based interface" which is being used over the world. Moreover, since the user community of Caffe provides a lively activity, the latest research results around the world tend to be provided by the Caffe-based implementation.

On the other hand, there is an attempt to achieve CNN on the mobile devices such as smartphones. Using the CNN in a mobile environment requires a high computing power to the device due to convolutional operations in the calculation process. When a CNN recognizes one image, billions of computations consisting of multiplication and summing are needed in general. In addition, there are also a problem related to mobile-specific memory capacity due to a large parameter files after training by using a framework such as Caffe. Especially, when we use the OSS framework for CNN, we can take advantage of the latest research results have been published. In order to use the learned parameter files by Caffe on the mobile, it is necessary to currently use Caffe for iOS/Android or the OpenCV DNN class. However, they are not optimized for the mobile devices and their execution speed is relatively slow.

In this study, we create a Caffe2C converter which converts the CNN model definition files and the parameter files trained by Caffe to a single C language code that can run on mobile devices. Since Caffe2C generates a single C code which includes everything needed to execute the trained CNN, Caffe2C makes it easy to use deep learning on the C language operating environment. Moreover, Caffe2C achieves faster execution speed in comparison to the existing Caffe for IOS/Android and OpenCV DNN class. The reasons are as follows: (1) directly converting of the Deep Neural Network to the C code, (2) efficient use of NEON/BLAS with multi-threading, (3) performing pre-computation as much as possible in the computation of the CNN.

In order to demonstrate the utilization of the Caffe2C, we have implemented four kinds of mobile CNN-based im-

MOBIQUITOUS '16 Adjunct Proceedings, November 28-December 01, 2016, Hiroshima, Japan

DOI: http://dx.doi.org/10.1145/3004010.3004025

age recognition apps on iOS. Thus, we explain the flow of construction of recognition app using Caffe2C.

To summarize it, our contributions in this paper are as follows:

- We create a Caffe2C converter which converts the model definition file and the parameter files of Caffe into a single C code that can run on mobile devices.
- We explain the flow of construction of recognition app using Caffe2C.
- We have implemented four kinds of mobile CNN-based image recognition apps on iOS.

2. RELATED WORK

To use Deep Convolutional Neural Network (CNN) efficiently on mobile devices the resources of which are limited such as computational power and memory size, CNN on mobile devices is actively being researched. [4, 8, 14] are works on the ingenuity of the calculation of the convolution layer that required the time of the majority of the processing.

Gong et al. [4] proposed to study a series of vector quantization methods for compressing the parameters. For the 1000-category classification task in the ImageNet challenge, they are able to achieve 16-24 times compression of the network with only 1% loss of classification accuracy using the state-of-the-art CNN.

Liu et al. [14] proposed efficient sparse matrix multiplication algorithm on CPU for Sparse Convolutional Neural Networks (SCNN) models. They apply the SCNN model to the object detection problem, in conjunction with a cascade model and sparse fully connected layers, to achieve significant speed-up.

Jade et al. [8] proposed a method approximating a learned full rank filter bank as combinations of a rank-1 filter basis. They demonstrate this method for scene text character recognition, showing a possible 2.5× speedup with no loss in accuracy, and 4.5× speedup with less than 1% drop in accuracy, still achieving state-of-the-art on standard benchmarks.

In addition, [3, 13] is a research of acceleration focusing on the weight parameters according to the training.

Courbariaux et al. [3] proposed a method called "BinaryConnect" which consists in training a DNN with binary weights during the forward and backward propagations, while retaining precision of the stored weights in which gradients are accumulated.

Lin et al. [13] proposed a method called "quantized back propagation" which converts multiplications into bit-shifts. They show better performance than the previous studies of BinaryConnect.

As mentioned above, there are majority studies related to accelerate CNN. However, our recognition engine is sufficiently fast. Therefore, we have determined that there is no need of additional contrivance for our recognition engine, especially, no need for accelerating recognition engine particularly. In addition, we use Network-In-Network(NIN) which has no fully connected layer which accounts for 90% of the parameters of the total number of parameters. Without parameter compression, the model size was 22MB which we regarded as moderate size for mobile implementation, we have determined that there is no need parameter compression.

3. CONSTRUCTION OF CNN-BASED MOBILE IMAGE RECOGNITION SYSTEM

In this chapter, we explain Caffe2C which converts the model definition file and the parameter files of Caffe to a single C language code that can run on mobile devices. Moreover, we explain the flow of CNN-based mobile image recognition application using Caffe2C.

3.1 Caffe2C

Caffe is the most common CNN framework which was released at the earliest CNN rise. Although the parameter files are generated after training using Caffe, it can not be directly used in mobile devices. In order to use the learned parameters by Caffe on mobile devices, it is necessary to currently use Caffe for iOS/Android or the OpenCV DNN class. However, they are not so efficiently optimized for mobile devices that the execution speed is relatively slow.

In this study, we create a Caffe2C converter which converts the model definition files and the parameter files into a single C language code that can run on mobile devices. Since Caffe2C generates a single C code which includes everything needed to execute the trained CNN, Caffe2C makes it easy to use deep learning on the C language operating environment. In the generated C code by Caffe2C, the CNN model is represented as a C source code, which removes pursing of the model definition files, and all the trained parameters are embedded in the C source code as constant arrays. The generate code achieves faster execution speed in comparison to the existing Caffe for IOS/Android and the OpenCV DNN class. The reasons are as follows:

1. Caffe2C directly converts the Deep Neural Network to a C source code.
2. Caffe2C effectively uses NEON/BLAS by multi-threading.
3. Caffe2C performs the pre-computation of the CNN as much as possible to reduce the amount of online computation.

Therefore, we can extend the applicability of CNN by using the Caffe2C. To train a CNN model in Caffe, we can use a DIGITS which is a Web-based interface of Caffe provides a simple and easy solution to training a CNN model. Hence, using both DIGITS for training CNN and Caffe2C for conversion of model and parameter files make it easy to use deep learning on mobile devices. In the next section, we explain the flow of construction of recognition app using Caffe2C.

3.2 Flow of Making Recognition Mobile Application

The flow of making recognition mobile app is as follows. From Step 1 to Step 5, if these steps are only once created, it is possible to use repeatedly. Step 2 can easily train the dataset if you use the DIGITS instead of Caffe framework. Thus, if only Step 2 would be prepared, we can quickly and easily create any recognition system on the mobile devices. In addition, general versatility of implementation is high potential capabilities because the parameter files that are automatically generated by the Caffe2C.

flow of making recognition mobile app

1. Prepare a training image data
2. Train a CNN model by Caffe (or DIGITS)
3. Generate a C source code by Caffe2C automatically
4. Prepare a GUI code of mobile app
5. Generate CNN-based image recognition app by compiling the generated C code and the GUI code

4. IMAGE RECOGNITION ENGINE

In this section, we explain a recognition engine used in the mobile CNN-based image recognition applications in this paper.

4.1 Basic recognition CNN architecture

As standard architectures of CNN for image recognition, AlexNet [11], GoogleNet [18], VGG-16 [17], and Residual-Net [5] are commonly used. However, mobile CNN implementation is limited in terms of application capacity such as memory and computational costs. Therefore, we select the Network-In-Network(NIN) [12] as a basic CNN architecture for our recognition engine. NIN has no fully connected layers which needs a huge number of parameters in other CNNs such as AlexNet and VGG-16. Figure 1 shows the NIN architecture. Note that Caffe2C is not limited to only NIN and can convert any kinds of CNNs into a C code.

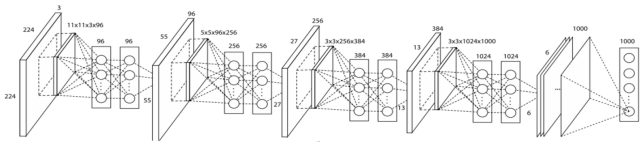

Figure 1: Network-In-Network(quoted from [12]). Composed of only Conv layer. Adding a BN [6] just before the ReLU function of all layers.

Thus, the number of the total large-scale parameters of AlexNet are about 62 million, but NIN are about 7.6million which is relatively smaller than the others since NIN has no fully connected layers. It has achieved a significant reduction of the number of parameters by using the NIN neural network architecture. In addition, in this paper, we add batch normalization (BN) [6] layers just after all the convolutional (Conv) layers and cascaded cross channel parametric pooling (CCCP) layers to the NIN architecture for accelerating deep neural network training. Thus, recognition performance is maintained at the same level of recognition performance of AlexNet in 1000 class recognition. Table 1 shows the recognition performance of 1000 class. In view of the mobile implementation, we consider that NIN architecture is reasonable.

Using the NIN model, we pre-trained CNN with ImageNet 2000 class images (totally 2.1 million images) which consisted of ILSVRC2012 1000 class images and 1000 food-related class selected from all the 21,000 ImageNet class. Further, we fine-tuning CNN with each recognition target dataset(food, general object 2000, bird, dog, etc.). Moreover, we use the Caffe [9] in training.

Finally, we convert training models at Caffe to an operable C code in the iOS/Android environment by Caffe2C.

Table 1: Recognition performance of 1000 class in AlexNet and NIN

	param	memory	top-5
AlexNet	62million	248MB	95.1%
NIN(4 layer +BN)	5.5million	22MB	95.2 %
NIN(5 layer +BN)	15.8million	63MB	96.2 %

4.2 CNN acceleration method

To transfer the convolutional operation to matrix product, we use the Im2col(image-to-column) operation which is effectively a 3D array, into a 2D array that we can treat like a matrix. Figure 2 shows visualizing Im2col operation.

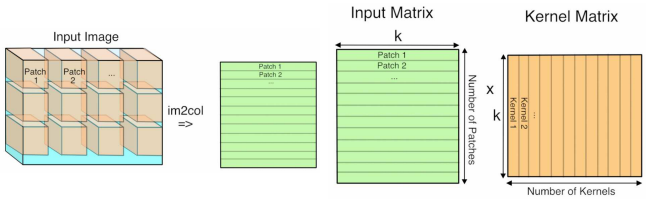

Figure 2: Transfer convolutional operation to GEMM

In the calculation of CNN, GEneral Matrix to Matrix Multiplication(GEMM) are frequently used, therefore, we use the BLAS implementation of accelerate framework which is highly optimized for the device in iOS. On the other hand, we use the BLAS implementation of OpenBLAS in Android.

Moreover, We create your own GEMM routine using the NEON instruction. The NEON is a Single Instruction Multiple Data (SIMD) instruction set of the ARM processor that is the vector processing system that enables a single instruction to process multiple pieces of data. The NEON instructions calculate the four 32bit single-precision floating-point at the same time. In addition, by the multi-processor, it is possible to run a total of eight operations of iOS in two CPU cores concurrency at the same time. Further, the number of cores in the Android is generally four CPU cores, therefore, total of 16 operations can run concurrently, it is possible to accelerate the convolutional operation.

Note that the code for Im2col and GEMM operation are includes in the C source code automatically generated by Caffe2C.

In summary, acceleration method is as follows:

- BLAS implementation
  (Accelerate Framework for iOS/OpenBLAS for Android)
- NEON instruction (same both iOS and Android)

5. EXPERIMENTS

We performed the measurement of recognition time on the iOS/Android. We evaluated the recognition time of both the engine accelerated by BLAS and NEON as well.

5.1 Experimental settings

The recognition engine was speeded up by BLAS and NEON implemented on the iOS and Android. We evaluated the recognition time on the mobile devices. Experimental settings are as follows:

- Implementation on the iOS accelerated by BLAS
- Implementation on the iOS accelerated by NEON
- Implementation on the Android accelerated by BLAS
- Implementation on the Android accelerated by NEON

5.2 Devices for evaluation

The used devices for evaluation are as follows:

- iPhone 7 Plus (CPU A10 2.33GHz 3GB DualCore iOS10)
- iPad Pro (CPU A9X 2.25GHz RAM4GB DualCore iOS10)
- iPhone SE (CPU A9 1.85GHz RAM2GB DualCore iOS10)
- GALAXY Note 3 (2.3GHz RAM3GB QuadCore Android5.0)

5.3 Results

Table 2 shows average recognition time each implementation. We repeated recognition 20 times and averaged the measured time. As a result, we revealed that BLAS was better for iOS, while NEON was better for Android.

Table 2: Recognition time[ms]

	NEON	BLAS	devices	BLAS
iOS	181.0	55.7	iPhone 7 Plus	Accelerate
iOS	222.4	66.0	iPad Pro	Accelerate
iOS	251.8	79.9	iPhone SE	Accelerate
Android	251	1652	GALAXY Note 3	OpenBLAS

6. MOBILE APPLICATIONS

In order to demonstrate the utilization of the Caffe2C, we implemented four kinds of mobile CNN-based image recognition apps on iOS. In this section, we explain six kinds of mobile recognition app, DeepFoodCam, DeepBirdCam, DeepDogCam, and DeepFlowerCam. The food recognition app, DeepFoodCam, works on both Android/iOS, while the others are implemented on only iOS.

6.1 DeepFoodCam

Figure 3 shows 101 class food recognition app, DeepFoodCam. Table 3 shows classification accuracy. This app can recognize 101 classes including 100 food classes and one non-food class. In the training time, we fine-tuned the CNN with 101 class images (totally 20,000 images) which consisted of UECFOOD-100 [15, 21] class images and non-food images collected from Twitter.

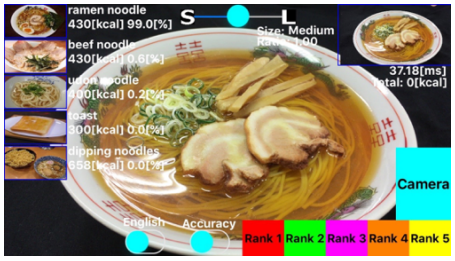

Figure 3: The screen-shot of DeepFoodCam. "Ramen noodle" was recognized.

Table 3: Classification accuracy of 101 food classes. Note that Top-N means the ratio of that the top-N candidates include the true classes.

	top-1	top-5
food 101 class	74.5%	93.5%

6.2 DeepBirdCam

Figure 4 shows bird 200 class recognition app, DeepFoodCam. Table 4 shows classification accuracy. This app can recognize 200 bird classes. In training, we fine-tuning CNN with 6033 images of Caltech-UCSD Birds 200 Dataset [20]. For evaluation, we used 25 % images, while we used the rest 75% images for training.

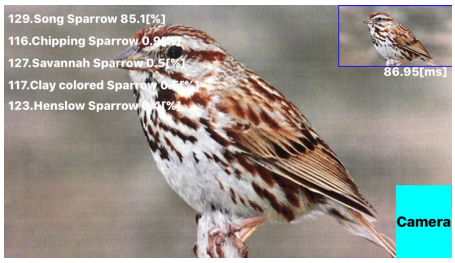

Figure 4: The screen-shot of DeepBirdCam. "Song sparrow" was recognized.

Table 4: Classification accuracy of 200 bird classes.

	top-1	top-5
bird 200 class	55.8%	80.2%

6.3 DeepDogCam

Figure 5 shows dog 100 class recognition app, DeepDogCam. Table 5 shows classification accuracy. This app can recognize 100 dog classes. In training, we fine-tuning CNN with 150 and over images per class of Stanford Dogs Dataset Dataset [10]. For evaluation, we used 25 % images.

Figure 5: The screen-shot of DeepDogCam. "Maltese dog" was recognized.

Table 5: Classification accuracy of 100 dog classes.

	top-1	top-5
dog 100 class	69.0%	91.6%

6.4 DeepFlowerCam

Figure 6 shows flower 102 class recognition app, DeepFlowerCam. Table 6 shows classification accuracy. This app can recognize 102 flower classes. In training, we fine-tuning CNN with 80 and over images per class of 102 Category Flower Dataset [16]. For evaluation, we used 25 % images.

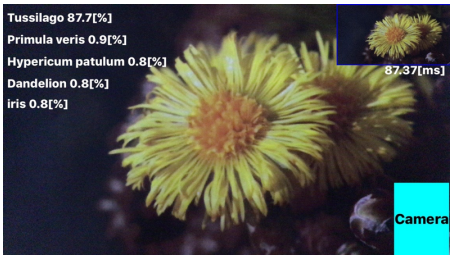

Figure 6: The screen-shot of DeepFlowerCam. "Tussilago" was recognized.

Table 6: Classification accuracy of 102 flower classes.

	top-1	top-5
flower 102 class	64.1%	85.8%

7. CONCLUSIONS

Caffe is widely used as a deep learning framework over the world. In order to use the learned parameters by Caffe on mobile devices, it is necessary to currently use Caffe for the iOS/Android or the OpenCV DNN class. However, they are not optimized for the mobile devices, and their execution speed is relatively slow.

In this study, we create the Caffe2C converter to convert the parameter files trained with Caffe to the C language code that can be run on the mobile devices. Caffe2C achieved fast execution speed in comparison to the existing Caffe for IOS/Android and the OpenCV DNN class. In addition, we have demonstrated the availability of Caffe2C through the flow of making recognition mobile application using Caffe2C and its automatically generated recognition codes.

For future work, we plan to apply our mobile framework into real-time CNN-based mobile image processing such as Neural Style Transfer [7]. Thereby, we plan to also create a converter for other frameworks such as Chainer [19].

8. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. http://tensorflow.org/.

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proc. of the Python for Scientific Computing Conference (SciPy)*, 2010.

[3] M. Courbariaux, Y. Bengio, and J. P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 2015.

[4] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. In *Proc. of International Conference on Learning Representations*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of arXiv:1512.03385*, 2015.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[7] A. A. J. Johnson and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. of arXiv:1603.08155*, 2016.

[8] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. of arXiv: 1405.3866*, 2014.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the ACM International Conference on Multimedia*, pages 675–678, 2014.

[10] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[12] M. Lin, Q. Chen, and S. Yan. Network in network. In *Proc. of International Conference on Learning Representations*, 2014.

[13] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. In *Proc. of arXiv:1510.03009*, 2015.

[14] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proc. of IEEE Computer Vision and Pattern Recognition*, 2015.

[15] Y. Matsuda, H. Hoashi, and K. Yanai. Recognition of multiple-food images by detecting candidate regions. In *Proc. of IEEE International Conference on Multimedia and Expo*, 2014.

[16] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proc. of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman. International conference on learning represenation workshop track. In *Deep inside convolutional networks: Visualising image classification models and saliency maps*, 2014.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc.of IEEE Computer Vision and Pattern Recognition*, 2015.

[19] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning, 2015. https://github.com/pfnet/chainer.

[20] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-ucsd birds 200. Technical report, California Institute of Technology, 2010.

[21] K. Yanai and Y. Kawano. Food image recognition using deep convolutional network with pre-training and fine-tuining. In *Proc. of ICME Workshop on Multimedia for Cooking and eating Activities (CEA)*, 2015.