Efficient Mobile Implementation of A CNN-based Object Recognition System

Keiji Yanai Ryosuke Tanno Koichi Okamoto
Department of Informatics,
The University of Electro-Communications, Tokyo
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585 JAPAN
{yanai,tanno-r,okamoto-k}@mm.inf.uec.ac.jp

ABSTRACT

Because of the recent progress on deep learning studies, Convolutional Neural Network (CNN) based method have outperformed conventional object recognition methods with a large margin. However, it requires much more memory and computational costs compared to the conventional methods. Therefore, it is not easy to implement a CNN-based object recognition system on a mobile device where memory and computational power are limited.

In this paper, we examine CNN architectures which are suitable for mobile implementation, and propose multi-scale network-in-networks (NIN) in which users can adjust the trade-off between recognition time and accuracy. We implemented multi-threaded mobile applications on both iOS and Android employing either NEON SIMD instructions or the BLAS library for fast computation of convolutional layers, and compared them in terms of recognition time on mobile devices. As results, it has been revealed that BLAS is better for iOS, while NEON is better for Android, and that reducing the size of an input image by resizing is very effective for speedup of CNN-based recognition.

Keywords

convolutional neural network, mobile implementation, network-in-network, iOS, Android

1. INTRODUCTION

Due to the recent progress of the studies on deep learning, convolutional neural network (CNN) based methods have outperformed conventional methods with a large margin. Therefore, CNN-based recognition should be introduced into mobile object recognition. However, since CNN computation is usually performed on GPU-equipped PCs, it is not easy for mobile devices where memory and computational power is limited.

In this paper, we explore the possibility of CNN-based object recognition on mobile devices, especially on iOS and Android devices including smartphones and tablets in terms of processing speed and required memory.

MM '16, October 15 - 19, 2016, Amsterdam, Netherlands

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3603-1/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2964284.2967243

First, we compare the CNN architectures for generic object recognition and select Network-in-Network (NIN) [1] as a basic architecture for our mobile CNN implementation. Next, we explore weight compression of NIN by Product Quantization (PQ) [2]. Regarding processing time, we describe fast mobile implementation and multi-scale recognition which allows users to adjust the trade-off between recognition time and accuracy. In addition, we modify the standard NIN architecture by introducing batch normalization (BN) [3] and adding layers to raise recognition performance. Finally we examine the above-mentioned points by the experiments.

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

- We compare basic CNN architectures and show the effectiveness of network-in-network for mobile implementation.
- We show multi-scale recognition of NIN is helpful to balance recognition time against accuracy.
- We have implemented NIN-based CNN recognition engines on both iOS and Android which can run in 32.6 ms in the fastest setting.

2. RELATED WORK

Recently the optional package of the OpenCV 3.1 library[1] distributed as "contrib" modules experimentally has supported a DNN module in which the network trained in Caffe [4] and Torch-7 [5] can be deployed. Since OpenCV supports both iOS and Android, anyone can implement a mobile deep learning system easily with the DNN module. However, no special optimization on the DNN module was carried out for mobile devices. To run AlexNet on iOS or Android it requires more than one second and much storage to store the trained weights. For example, 60 million weights are needed for AlexNet.

An open-source mobile deep learning software, DeepBeliefSDK[2] is optimized for Android and iOS where the BLAS library is employed on iOS and NEON SIMD instruction sets are used on Android. It took about 130 ms for one-time recognition on iPad Pro. In addition, to reduce memory to store the weights, one weight is represented in one byte by quantization [6]. Its drawback is that it is difficult to change CNN models from the built-in AlexNet to the others, since training function or importing function from other frameworks is not provided.

As academic studies, many researchers try to compress weights and reduce computational costs. Gong et al. [7] proposed to apply Product Quantization (PQ) [2] to convnets

[1]http://opencv.org/
[2]https://github.com/jetpacapp/DeepBeliefSDK

to compress their weights. However, they applied PQ-based compression to only fully connected (FC) layers in AlexNet and not to convolutional (conv) layers. Yang et al. [8] also proposed Fastfood transform and applied it for compression of FC layers in Alexnet, and Han et al. [9] proposed a method which combines pruning, quantization and Huffman Encoding, and applied it to both conv and FC layers in Alexnet and VGG-16. Courbariaux et al. [10] proposed a method to binarize network weights to reduce computational costs and required memory.

However, most of the authors did not implement their proposed method on actual mobile devices. Exceptionally, Wu et al. [11] implemented and tested Compressed AlexNet on an Android smartphone. However, it took 0.95 seconds for one-time recognition, which was much slower than our implementation.

In this paper, we aim to implement a practical CNN-based object recognition engine on both iOS and Android in terms of computational speed and required memory to store CNN weights.

Before the deep learning era got started, several local-feature-based mobile recognition systems were proposed. For example, Kawano *et al.* [12] proposed a mobile food recognition system which can classify 100 food classes in only 0.065 seconds employing Fisher Vector (FV) [13]. To realize fast recognition, they employed weight compression technique [14] and multi-threading implementation. This system was called “FoodCam” which was implemented as an interactive application taking advantage of their very quick recognition. One of our objectives in this paper is implementing such a fast mobile recognition system using a state-of-the-art CNN architecture. We propose “DeepFoodCam” which is a CNN-version of “FoodCam” as an example of CNN-based mobile recognition systems in the last part of this paper.

3. CNN ARCHITECTURE

As basic CNN architectures for object recognition, AlexNet [15], Network-in-Network (NIN) [1], GoogLeNet [16] and VGG-16 [17] are commonly used. For comparison, we show the basic characteristics of their CNN architecture regarding the number of layers, weights and computation on convolutional (conv) layers and fully-connected (FC) layers in Table 1 [3].

AlexNet is the first large-scale object recognition CNN which consists of 6 conv layers and 3 FC layers, while VGG-16 is a deeper version of AlexNet which has 13 conv layers and 3 FC layers. FC layers need a large number of weights, 59M for AlexNet and 129M for VGG-16. Although GoogLeNet also has a deep architecture consisting of 21 conv layers, it has only one FC layers with 1M parameters. While VGG-16 is a deep and high-performance but simple architecture which uses only 3×3 conv layers, GoogLeNet is quite complicated which consists of many “Inception modules” each of which consists of 5x5, 3x3, 1x1 conv layers and pooling layers.

For mobile implementation, these three architectures are not appropriate, because AlexNet and VGG-16 have too much parameters and GoogLeNet is a complicated architecture which is difficult for effective parallel implementation for mobile devices. Unlike these three architectures, Network-in-Network (NIN) has a simple architecture consisting of no FC layers and 12 conv layers. Instead of FC layers, NIN adopts Cascaded Cross Channel Parametric Pooling (CCCP Pooling) which is implemented by two consecutive 1 conv layers just after 3×3 or larger conv layers. The number of the total parameters of NIN are 7.6 million, and the number of the multiplication is 1.1 billion, which is relatively smaller than the other architectures.

[3]We referred to the open models at Caffe Model Zoo (https://github.com/BVLC/caffe/wiki/Model-Zoo).

Table 1: Comparison of CNN Architectures regarding the number of layers, weights and computation (multiplication). “ImageNet” means the top-5 error rate for the ImageNet Challenge data with a single model.

model		Alex	VGG-16	GoogLeNet	NIN
conv	layer	5	13	21	12
	weights	3.8M	15M	5.8M	7.6M
	comp.	1.1B	15.3B	1.5B	1.1B
FC	layer	3	3	1	0
	weights	59M	124M	1M	0
	comp.	59M	124M	1M	0
TOTAL	weights	62M	138M	6.8M	7.6M
	comp.	1.1B	15.5B	1.5B	1.1B
ImageNet	top-5 err.	17.0%	7.3%	7.9%	10.9%

Given these characteristics of NIN, we have decided to adopt NIN as a basic architecture of mobile implementation in this work. In addition, NIN can accept an image of any size as an input of the network by using global pooling instead of fixed-size pooling in the last pooling layer, because NIN is a CNN without FC layers. This is very helpful for mobile implementation, because users can adjust the balance between recognition speed and accuracy by their preference without changing the network weights. We will refer this point later.

4. COMPRESSION OF NIN WEIGHTS

In this paper, we adopt Network-in-Network (NIN) [1] as a basic CNN architecture, because both the amount of the parameters and computational costs are relatively smaller. However, 30 MByte at least is still needed to store 7.6 million weights in 32-bit single floating point representation. Then we introduce weight compression.

Inspired by Gong et al. [7], we apply product quantization (PQ) [2] for weight compression of conv layers of NIN, although Gong et al. applied PQ-based compression not to conv layers but only to FC layers in AlexNet. By the experiments, we confirm the effectiveness of PQ-based weight compression for conv layers of NIN.

5. FAST IMPLEMENTATION ON MOBILE DEVICES

In this work, we focus on iOS and Android devices including smartphones and tablets as target devices for mobile implementation.

For fast mobile implementation, it is essential to use (1) Multi-threading, (2) SIMD instruction sets and (3) iOS Accelerate Framework (only for iOS). In addition, it is possible to reduce computational costs by devising CNN architectures, computation algorithms, and introducing approximation into CNN computation. As the other easier way to reduce computational costs, we can reduce the size of an input image.

In this section, we describe (1), (2) and (3) which we consider are prerequisite for fast implementation of CNNs on mobile devices. Since so many works have been published regarding reducing computational costs of CNNs as mentioned in Sec.2, we like to keep introducing them for our future work. Instead, to verify the effectiveness of reducing

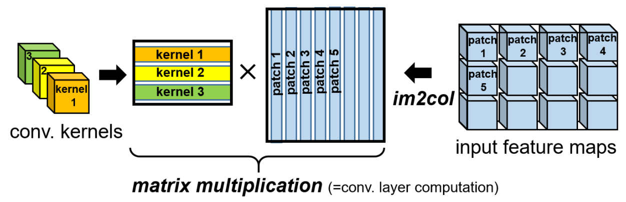

Figure 1: Computation of conv layers with "im2col" and "matrix multiplication".

of the size of an input image for speedup, we will examine the relations between input image sizes and recognition accuracy in the next section.

Because NIN has only conv layers as the layers having trained parameters, we need efficient mobile implementation of feed-forward computation of convolutional layers. In general, computation of conv layers is decomposed into "im2col" operation and matrix multiplications. In im2col operation, each patch to apply a conv kernel to is extracted from a given feature map, and place them in one column of a patch matrix, while each conv kernel is also placed in one row of a kernel matrix (See Figure 1). After that, we can obtain the output feature maps of the target conv layer by multiplying the kernel matrix by the im2col patch matrix. In general, this generic matrix multiplication (GEMM) is very time-consuming, since both matrices are large (sometime larger than 1000×1000). Although we can use GPU for fast GEMM calculation in case of PC or servers, for mobile devices we need to use CPU as effective as possible.

(1) Multi-threading: Most of the recent CPU on mobile devices have four CPU cores. By using four cores effectively, four times speedup can be expected. In our implementation, to use four CPU cores by multi-threading, we divide a kernel matrix into four sub-matrices along the row. As shown in Table 1, the number of conv kernels are usually multiples of four. Four GEMM operations are carried out in parallel. Note that most of the CPUs on iOS devices including the latest APPLE CPU, A9X, have only two cores. Therefore, only twice speedup is expected with multi-threading for iOS devices at most.

(2) SIMD instruction: Most of the recent mobile devices are equipped with ARM-based CPUs except for some devices having Intel Atom. Therefore, we focus only on NEON which is a SIMD instruction set of ARM processors. Among the NEON instruction set, we use "*vmla.f32*" which can execute four multiplications and accumulating calculations at once. This operations are frequently used for GEMM calculation. NEON instructions can be carried out in each CPU core independently. Therefore, by combining multi-threading and NEON, 16 times speedup for Android and 8 times for iOS are expected at most.

(3) iOS Accelerate Framework: In case of iOS, iOS Accelerate Framework which includes highly optimized BLAS (Basic Linear Algebra Subprograms) library is available. The GEMM function in the BLAS library ("*cblas_sgemm*") can be used for GEMM operations. BLAS in iOS probably uses NEON instructions inside the library. However, the detail is not disclosed. (2) and (3) are alternative to each other. We will compare them in the experiments.

Table 2: The original and variants of the NIN architectures.

layer no.	(1) original NIN	(2) 4layers + BN	(3) 5layers + BN
1	11x11x96 **conv1**	11x11x96 **conv1**	**11x11x96 conv1**
2	1x1x96 cccp1_1	1x1x96 cccp1_1	1x1x96 cccp1_1
3	1x1x96 cccp1_2	1x1x96 cccp1_2	1x1x96 cccp1_1
4	5x5x256 **conv2**	3x3x256 **conv2_1**	3x3x256 **conv2_1**
5	1x1x256 cccp2_1	3x3x256 **conv2_2**	3x3x256 **conv2_2**
6	1x1x256 cccp2_2	1x1x256 cccp2_1	1x1x256 cccp2_1
7	3x3x384 **conv3**	1x1x256 cccp2_2	1x1x256 cccp2_2
8	1x1x384 cccp3_1	3x3x384 **conv3**	3x3x384 **conv3**
9	1x1x384 cccp3_2	1x1x384 cccp3_1	1x1x384 cccp3_1
10	3x3x1024 **conv4**	1x1x384 cccp3_2	1x1x384 cccp3_2
11	1x1x1024 cccp4_1	3x3x768 **conv4**	3x3x768 **conv4**
12	1x1xN cccp4_2	1x1x768 cccp4_1	1x1x768 cccp4_1
13	avg. pool	1x1xN cccp4_2	1x1x768 cccp4_2
14	sofmax	avg. pool	3x3x1024 **conv5**
15		softmax	1x1x1024 cccp5_1
16			1x1x1024 cccp4_2
17			avg. poling
18			softmax
weights	7.6Million	5.5Million	15.8Million
computation	1.1Billion	1.2Billion	1.7Billion

6. MULTI-SCALE RECOGNITION FOR ADJUSTING TIME AND ACCURACY

The original NIN is designed so that the size of an input image of NIN is 227×227. However, by using global average pooling which accepts feature maps of any size in the final pooling layer, NIN can accept input images of any size, since NIN has no FC layers. This is a helpful characteristics for mobile implementation, since users can adjust processing speed by changing the size of an input image without changing CNN weights. The smaller the size of an input image becomes, the faster the NIN can recognize it. Instead, the accuracy will be degraded.

In the experiments, we examine the trade-off between recognition time and accuracy with input images of several sizes and two kinds of methods to generate smaller images. One is cropping smaller images from the original one without resizing, while the other is resizing original one to smaller images.

7. EXTENSION OF THE NIN ARCHITECTURE

The original NIN consists of four repetitive small structures each of which includes one conv layer and Cascaded Cross Channel Parametric Pooling (CCCP) layers which are equivalent to two consecutive 1×1 conv layers. When training, we added drop-out [18] to the two last 3×3 conv layers to make training easier.

Recently batch normalization (BN) [3] has been proposed, which can accelerate training without drop-out and make training of deeper networks possible. Then, we build two variant models of NIN by adding BN layers just after all the conv/cccp layers and one repetitive structures. One is "4-layers + BN" and the other is "5-layers + BN". In addition, we replaced 5x5 conv with two 3x3 conv layers, and reduced the number of kernels in conv_4 from 1024 to 768. The details of the original NIN and two variants are shown in Table 2. We will compare them regarding recognition performance in the experiments

Note that ReLUs (Rectified Linear Unit) exists just after all the conv/cccp layers, and in case of (2) and (3), BN (batch normalization) layers also exists just after all the conv/cccp layers and before ReLU, although they are not shown in the table.

Table 3: Recognition time [ms] on mobile devices.

	NIN(BLAS)	NIN(NEON)	NIN4	NIN5	D-Belief
iPad Pro	**66.0**	221.4	66.6	103.5	131.9
iPhone SE	**79.9**	251.8	77.6	116.6	137.7
Galaxy Note 3	1652	**251.1**	-	-	-

Table 4: Recognition accuracy (top-1/5) [%] of the trained models.

	ImageNet2000		UEC-FOOD		
model	top-1	top-5	top-1	top-5	weights
FV(HOG+color)[12]	-	-	65.3	86.7	5.6M
AlexNet	44.5	67.8	78.8	95.2	62M
NIN	41.9	65.9	75.0	93.7	7.6M
NIN(4layers+BN)	39.8	65.0	77.9	94.6	5.5M
NIN(5layers+BN)	45.8	70.5	80.8	95.4	15.8M

8. EXPERIMENTS

8.1 Implementation and Training

We have implemented a mobile deep learning framework which works on both iOS and Android. The framework supports only deployment of trained CNN models. We used Caffe [4] for training of CNN models on a PC with two Titan-X GPUs, and converted trained model files for our framework.

In the experiments, we mainly used the augmented UEC-FOOD100 dataset [19, 20] which is a 100-class food categories dataset containing at most 1000 food photos for each food class, because our objective is implementing practical CNN-based recognition engines. Following [20], before training with the food dataset, we pre-trained CNNs with ImageNet 2000 category images (totally 2.1 million images) which consisted of ILSVRC2012 1000 category images and 1000 food-related categories selected from all the 21,000 ImageNet categories.

For evaluation, we used 20% of ImageNet 2000 images and the images in the fold no.0 of the official split of UEC-FOOD100 as test images of ImageNet 2000 and augmented UEC-FOOD100, respectively. In the experiments, we measured processing time on mobile devices, while we evaluated recognition accuracy of the trained CNNs on a PC.

8.2 Recognition Time on Mobile Devices

We measured recognition times on iPad Pro 9.7inch (iOS 9.3), iPhone SE (iOS 9.3) and Galaxy Note3 (Android 5.0). We executed recognition more than twenty times and calculated the median of all the measured times for evaluation. Table 3 shows the processing time for one-time recognition with NIN with BLAS, NIN with NEON, NIN-4layers with BLAS, NIN-5layers with BLAS, and DeepBeliefSDK with BLAS (for comparison).

From these results, the BLAS library on iOS which was implemented as a part of iOS Accelerate Framework was very effective for speedup, while the OpenBLAS library which we used as BLAS implementation for Android was too slow. NEON implementation was moderate, although it was three times as slow as iOS BLAS implementation. For more speedup on Android, improvement of CNN models and computation such as reducing weights and approximation of CNN computation will be needed. This is one of our future works.

Although DeepBeliefSDK also employs BLAS, our NIN(BLAS) was twice as fast as it. One of the possible reason is that we used multi-threading, while they did not use.

For reference, we show the recognition accuracy for ImageNet2000 and UEC-FOOD with Fisher Vector (FV) based conventional method, AlexNet, NIN and two variants in Table 4. All the CNN based method outperformed the FV-based result [12] with a large margin. Both of two variants of NIN outperformed the original NIN except for NIN-4layers on ImageNet2000. Reducing the number of the kernels in the last conv layers for NIN-4layers is one of the possible reasons of this performance loss. NIN-5layers outperformed AlexNet, which showed that making a CNN deeper was effective for performance improvement.

Table 5: Time [ms] and top-1/5 accuracy [%] with images of various size.

(A) 4-layer + BN

Time	227x227		200x200		180x180		160x160	
iPad Pro	66.6		49.7		44.0		**32.6**	
iPhone SE	77.6		56.0		50.2		37.2	
Accuracy	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
resize	**78.8**	**95.2**	**77.3**	**95.1**	**76.0**	94.1	69.3	91.5
crop	**78.8**	95.2	75.8	93.9	72.0	92.1	63.0	87.7
multi-resize	74.7	93.9	74.0	94.6	74.4	**94.7**	**71.5**	**93.7**
multi-crop	74.7	93.9	70.8	92.2	69.8	92.2	61.4	87.2

(B) 5-layer + BN

Time	227x227		200x200		180x180		160x160	
iPad Pro	103.5		71.9		61.1		**46.6**	
iPhone SE	116.6		82.9		68.6		53.4	
Accuracy	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
resize	**81.5**	**96.2**	**80.2**	**95.7**	**78.4**	94.9	72.0	91.4
crop	**81.5**	**96.2**	78.3	95.1	75.1	93.6	65.3	87.3
multi-resize	78.2	95.3	78.2	95.1	78.2	**95.6**	**75.1**	**93.8**
multi-crop	78.2	95.3	75.8	93.2	73.1	92.2	66.3	88.3

Table 6: Accuracy (top-1/5) [%] with PQ-based weight compression.

	raw(32bit)	8bit	4bit	4bit(pair)	2bit(pair)
memory	30.4MB	7.6MB	3.8MB	3.8MB	1.9MB
top-1	75.0	74.5	66.8	72.9	50.3
top-5	93.7	93.5	89.7	92.9	78.1

8.3 Multi-scale Recognition

The advantage of NIN is that it can accept images of any size as input images. Table 5 shows relation between image size and time/accuracy. We used two methods to obtain smaller images, resizing and cropping. In addition, we trained NIN-4layers and NIN-5layers in multi-scale training where we resized training images with random magnification rate from 0.7 to 1.0 during training of the NIN models.

For the results, the processing time was proportional to the number of the pixels of an image. For example, in case of 160x160, the time was reduced by half. Although reducing the size from 227x227 to 180x180 brought only 2.8 point and 3.1 point accuracy loss for NIN-4layers and NIN-5layers, respectively, the processing time was reduced by two third.

As a method to reduce the size of an input image, resizing is much better than cropping in terms of accuracy.

The multi-scale trained models were not as effective as the normal models trained with 227x227 images except for 160x160 input images. The results by the multi-scale trained models are relatively uniform regardless the size of an input image, although the accuracy for the full-sized image was degraded compared to the normal models.

Until 180x180, reducing the size of an input image is effective and easy way to adjust the trade-off between accuracy and processing time.

8.4 Weight Compression

We applied Product Quantization (PQ) [2] to compress CNN weights for NIN to reduce required memory on mobile devices. Table 6 shows accuracy from no compres-

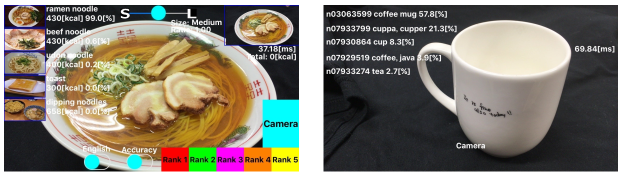

Figure 2: Screen-shots of food recognition app (left) and 2000-class recognition app (right).

sion to 1/16 compression. For "8bit" and "4bit", we applied quantization to each single element, while "4bit(pair)" and "2bit(pair)" means that we applied quantization to each pair of elements. Performance loss in case of "4bit(pair)" was only 2.1 point, although it brought 1/8 compression. From these results, PQ-based compression is helpful for NIN as well.

8.5 Mobile Applications

We have implemented two kinds of mobile CNN-based image recognition apps on iOS, a food recognition app, "DeepFoodCam", and a 2000-class object recognition app, which employ NIN trained with the augmented UEC-FOOD100 dataset including a additional non-food class, and NIN trained with 2000 ImageNet Food dataset, respectively. Both can recognize a 227x227 photo in 66ms and a 160x160 in 33ms.

We have released the iOS-version food recognition application on Apple App Store. Please search for "DeepFoodCam". The movies where two apps are working on iOS can be seen at our project page, http://foodcam.mobi/.

9. CONCLUSIONS

We proposed a mobile implementation of CNN which is based on Network-in-Network (NIN) [1]. For reducing the weights we examined PQ-based compression, while for fast recognition we adopted multi-threading and SIMD instruction or highly-optimized iOS Accelerate Framework for computation of convolutional layers. In addition, we examined NIN-based multi-scale recognition which enabled us to adjust the trade-off between time and speed. As results, it was turned out that reducing the size of input images was very effective. In addition, for a 160×160 input image, we achieved 32.6 ms recognition on iPad Pro, which is equivalent to the "real" real-time speed.

For future work, we plan to apply our mobile framework into real-time CNN-based mobile image processing such as super-resolution, semantic segmentation and neural style-transfer [21].

10. REFERENCES

[1] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. of International Conference on Learning Represenation Conference Track*, 2014.

[2] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of International Conference on Machine Learning*, pp. 448–456, 2015.

[4] 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 ACM International Conference Multimedia*, pp. 675–678, 2014.

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *Proc. of NIPS Workshop on BIGLearn*, 2011.

[6] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. of NIPS Workshop on Deep Learning and Unsupervised Feature Learnings*, 2011.

[7] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[8] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *Proc. of IEEE International Conference on Computer Vision*, pp. 1476–1483, 2015.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. of International Conference on Learning Representation*, 2016.

[10] 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*, pp. 3123–3131, 2015.

[11] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," *arXiv preprint arXiv:1512.06473*, 2015.

[12] Y. Kawano and K. Yanai, "Foodcam: A real-time food recognition system on a smartphone," *Multimedia Tools and Applications*, vol. 74, no. 14, pp. 5263–5287, 2015.

[13] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Proc. of European Conference on Computer Vision*, 2010.

[14] Y. Kawano and K. Yanai, "Ilsvrc on a smartphone," *IPSJ Transactions on Computer Vision and Applications*, vol. 6, pp. 83–87, 2014.

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

[16] 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*, pp. 1–9, 2015.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *Proc. of International Conference on Learning Represenation Workshop Track*, 2014.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] 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*, pp. 1554–1564, 2012.

[20] 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.

[21] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," *arXiv preprint arXiv:1603.08155*, 2016.