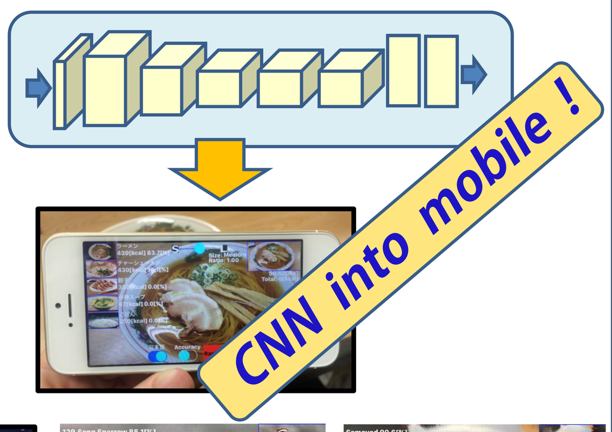
Efficient Mobile Implementation of A CNN-based Object Recognition System

Keiji Yanai, Ryosuke Tanno, Koichi Okamoto The University of Electro-communications, Tokyo, Japan

1. Objective

- ***Make it easier to implement CNN-based object recognition apps !***
  - High recognition accuracy by using Deep Convolutional Neural Network
  - Very high speed by efficient implementation
  - Memory saving by PQ-based weight compression
  - Converting CNN models trained by Caffe into mobile object recognition engines



- ***Example: 100-class food recognition***
  - recognition time: **26.2ms** (iPhone7+)
  - top-5 accuracy: 93.7%

2. Contributions

- ***Compare CNN architectures and select NIN for a base mobile CNN***
  - Compare AlexNet, VGG-16, GoogleNet and Network-in-Network (NIN)
  - Conclude that NIN is the best architecture for mobile implementation in terms of weight size and computational efficiency
- ***Examine fast CNN implementations on iOS and Android***
  - For iOS, using BLAS in the iOS Accelerate Framework is the best choice.
  - For Android, using NEON (SIMD instruction) is better than OpenBLAS.
- ***Adjust speed and accuracy by multi-scale NIN***
  - By introducing Global Average Pooling (GAP) into the last layer of NIN, NIN accepts input images of any size like FCN
  - User can adjust the balance between speed and accuracy by changing the size of input images
    e.g. iPhone 7 Plus: **26.2**[ms] for 160x160 imgs ⇔ **55.7**[ms] for 227x227
- ***PQ-based weight compression of Conv Layers***
  - 1/8 compression without significant performance loss

3. CNN Architectures & Multi-scale NIN

- ***As basic CNN architectures for object recognition, AlexNet, Network-in-Network (NIN), GoogLeNet and VGG- 16 are commonly used.***
  - The amounts of weights in **AlexNet** and **VGG-16** are too much for mobile.
  - **GoogLeNet** is too complicated for efficient parallel implementation. (It has many branches.)

model		AlexNet	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%

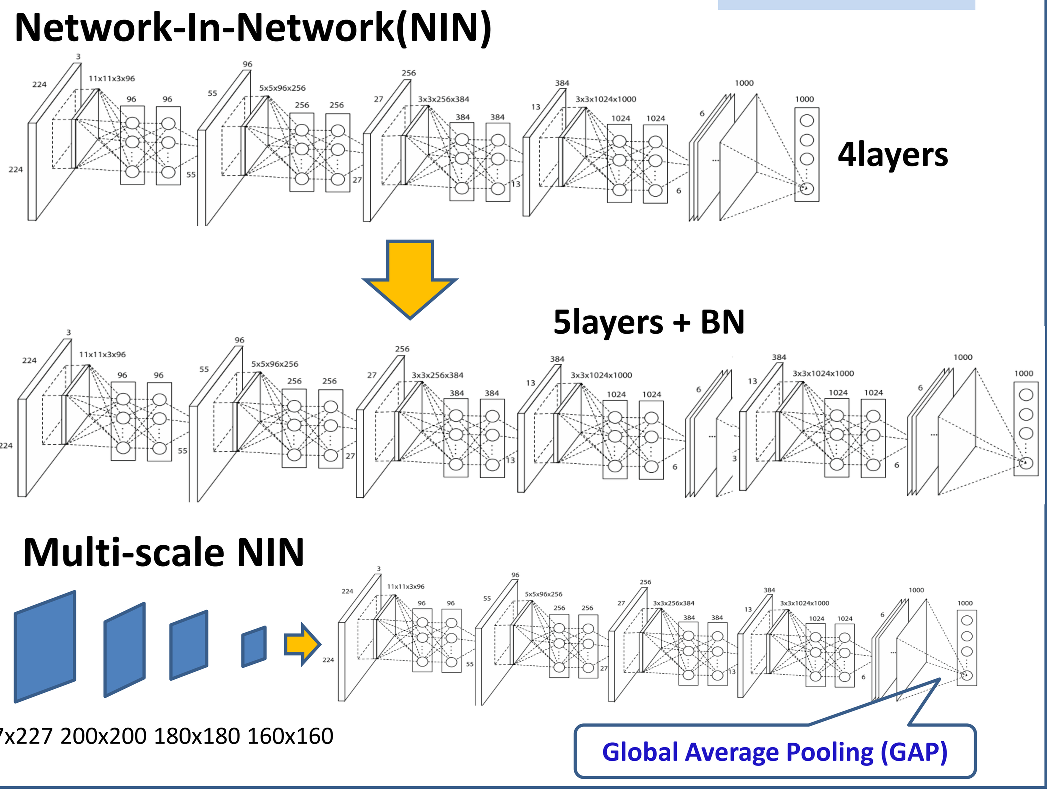
- ***We adopt Network-in-Network[1]***
  - No fully-connected layers (which brings less weights)
  - Straight flow and consisting of only convolutional and pooling layers

⇒ It's easy for parallel implementation. Efficient computation for convolutional layers is important !

- ***We modified models (BN, 5layer, multi-scale)***
  - adding BN[3] layers just after all the conv/cccp layers
  - replaced 5x5 conv with two 3x3 conv layers
  - reduced the number of kernels in conv 4 from 1024 to 768
  - **replaced fixed average pooling with Global Average Pooling**

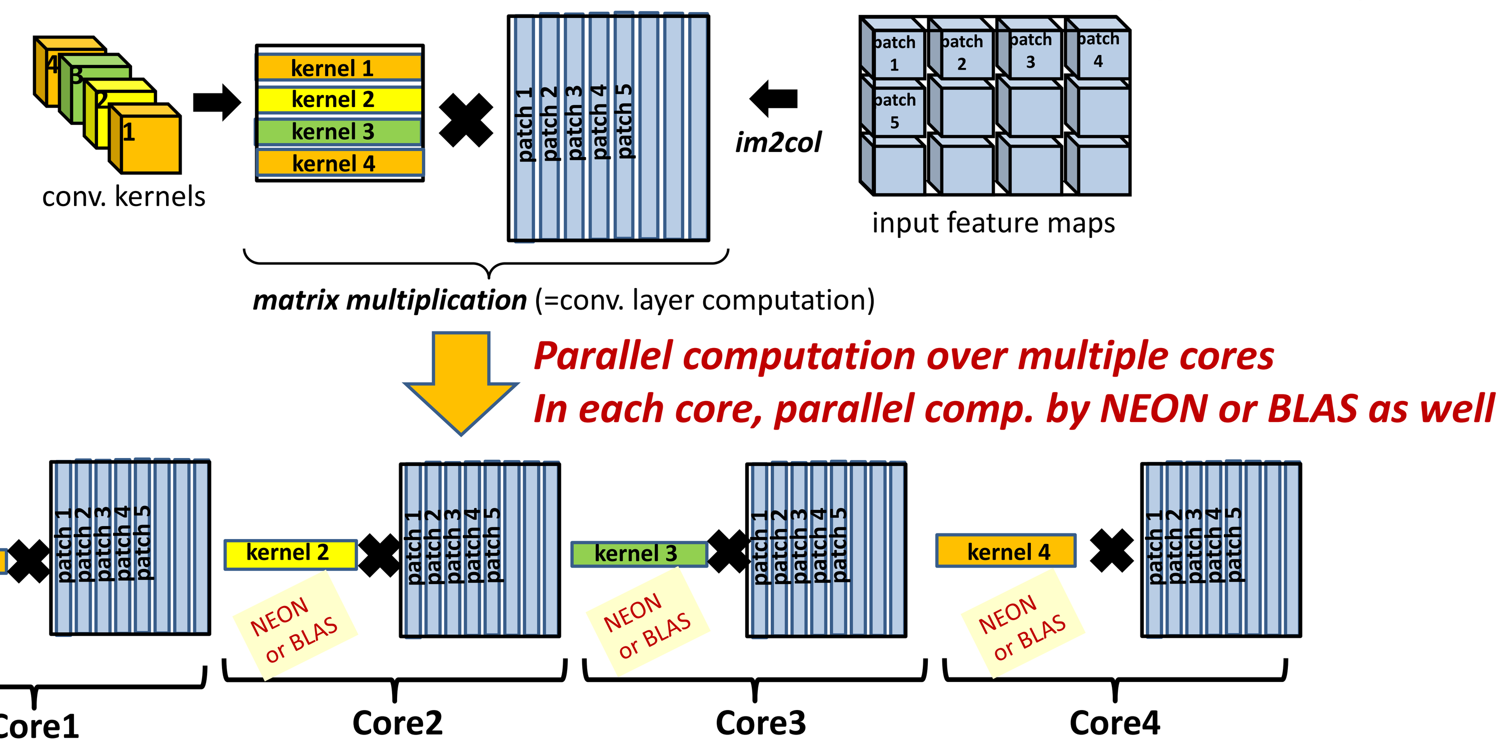
Trade-off: *Accuracy vs speed*
Ex. 4layer+BN (iPhone7Plus) (top-1)
227x227 55.7ms 78.8%
180x180 35.5ms 76.0%
160x160 26.2ms 71.5%

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_2
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	softmax	avg. pool	**3x3x1024** conv5
15		softmax	1x1x1024 cccp5_1
16			1x1xN cccp5_2
17			avg. pool
18			softmax
weights	7.6Million	5.5Million	15.8Million
computation	1.1Billion	1.2Billion	1.7Billion



4. Fast Implementation on Mobile Devices

- ***Speeding up Conv layers ⇒ Speeding up GEMM***
  - computation of conv layers is decomposed into "**im2col**" operation and generic matrix multiplications (**GEMM**)
  - ***Multi-threading : Use 2 cores in iOS , 4 cores in Android in parallel***
  - ***SIMD instruction (NEON in ARM-based processor)***
    NEON can compute four FP comp. in parallel. ⇒ four times speedup
    Total: iOS: 2Core*4 = 8 calculation, Android: 4Core*4 = 16 calculation
  - ***BLAS library (highly optimized for iOS ⇔ not optimized for Android)***
    BLAS (iOS: BLAS in iOS Accelerate Framework, Android: OpenBLAS)



5. Weight Compression

- ***We applied Product Quantization (PQ) [2] to compress CNN weights for NIN to reduce required memory on mobile devices.***
  - no compresion to 1/16 compression
  - 8bit and 4bit: We applied quantization to each single element
  - 4 bit(pair) and 2bit(pair): We applied quantization to each pair of elements
- ***PQ-based compression is helpful for NIN as well.***
  - Performance loss(4bit(pair)) was only 2.1 point, although it brought 1/8 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%

6. Experiments

- ***Implementation***
  - We have implemented a mobile deep learning framework which works on both iOS and Android.
  - Supports only deployment of trained CNN models on iOS and Android
  - Using Caffe for training of CNN models on a PC (2 Titan-X GPUs)

Recognition time on mobile devices (227x227)

	NIN(BLAS)	NIN(NEON)	NIN4	NIN5	D-Belief
iPhone 7 Plus	-	-	55.7[ms]	88.7[ms]	109.0[ms]
iPad Pro	**66.0**[ms]	221.4[ms]	66.6[ms]	103.5[ms]	131.9[ms]
iPhone SE	**79.9**[ms]	251.8[ms]	77.6[ms]	116.6[ms]	137.7[ms]
Galaxy Note 3	1652[ms]	**251.1**[ms]	-	-	-

- ***Training***
  - augmented UEC-FOOD 100 (1000 images / class)
  - Pre-trained CNNs with ImageNet 2000 category (ILSVRC2012 1000, food1000) (totally 2.1 million images)

Recognition accuracy of the trained models

model	ImageNet2000		UEC-FOOD		
	top-1	top-5	top-1	top-5	weights
FV(HOG+color)	-	-	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

Trade-off between Time and accuracy with images of various size for multi-scale NIN

(A)4-layer+BN

Time	227x227		200x200		180x180		160x160	
iPhone 7 Plus	55.7[ms]		42.1[ms]		35.5[ms]		**26.2**[ms]	
iPad Pro	66.6[ms]		49.7[ms]		44.0[ms]		32.6[ms]	
iPhone SE	77.6[ms]		56.0[ms]		50.2[ms]		37.2[ms]	
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	
iPhone 7 Plus	88.7[ms]		59.3[ms]		49.5[ms]		**38.7**[ms]	
iPad Pro	103.5[ms]		71.9[ms]		61.1[ms]		46.6[ms]	
iPhone SE	116.6[ms]		82.9[ms]		68.6[ms]		53.4[ms]	
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%

Reference

[1] M. Lin, and Q. Chen, and S. Yan.: Network In Network. Proc. of International Conference on Learning Representations, 2014
[2] H. Jegou, M. Douze, and C. Schmid.: Product quantization for nearest neighbor search, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011
[3] S. Ioffe and C. Szegedy.: Batch normalization: Accelerating deep network training by reducing internal covariate shift, Proc. of International Conference on Machine Learning, 2015