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

Ryosuke Tanno and Keiji Yanai

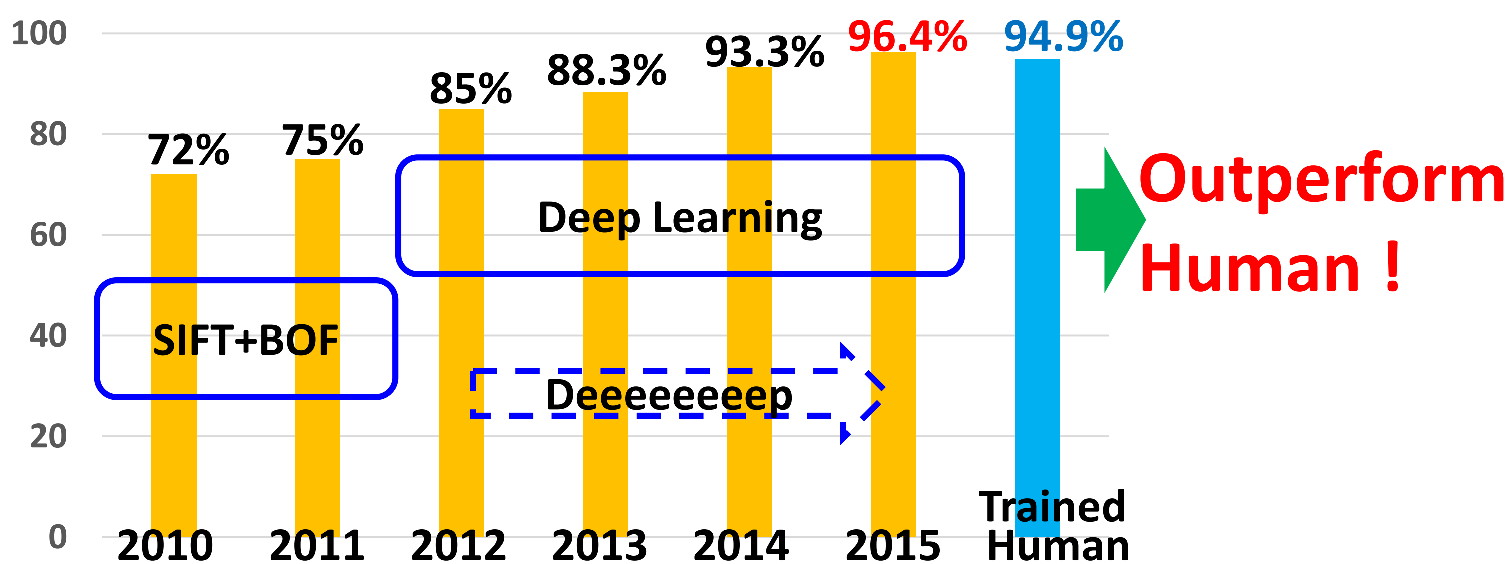
Department of Informatics,

The University of Electro-Communications, Tokyo

1. INTRODUCTION

Deep Learning(DNN,DCNN,CNN)

- Deep Learning achieved remarkable progress
  - E.g. Audio Recognition, Natural Language Processing,
- Especially, in Image Recognition, Deep Learning gave the best performance
  - Outperform even humans such as recognition of 1000 object(He+, Delving deep into rectifier, 2015)



Deep Learning Framework

- Many Deep Learning Framework have emerged
  - **E.g. Caffe**, TensorFlow, Chainer

What is Caffe?

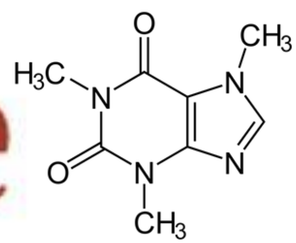
Convolution Architecture For Feature Extraction(CAFFE)

Open Framework, models and examples for Deep Learning

- Focus on Compuer Vision
- Pure C++/CUDA architecture for deep learning
- Command line, Python MATLAB interface
- Fastest processing speed
- **Caffe is the most popular framework in the world**

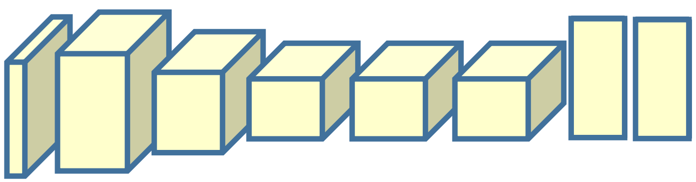
BVLC

Caffe



Bring to CNN to Mobile

- There are many attempts to archive CNN on the mobile
  - **Require a high computational power and memory**

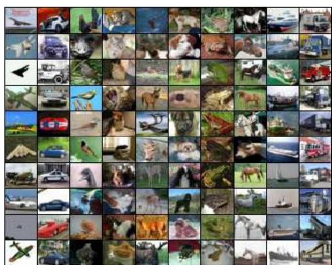


High Computational Power and **Memory** are **Bottleneck!!**

How to train a model by caffe?

- **3 files** are required for Training -> Output: **Model**
  - 3 files: **Network definition**, **Mean**, **Label**

Dataset



Files

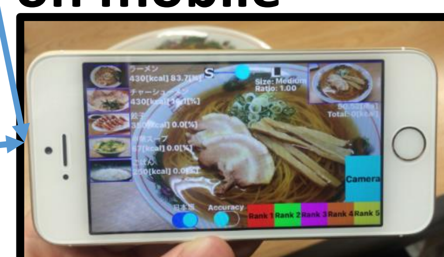
Training

Caffe

Output

- Caffemodel

Use these 4 files on mobile



- Network
- Mean
- Label

3 files

Use the 4 Files by Caffe on the Mobile

- We currently need to use OpenCV DNN module
  - **not optimized** for the mobile devices
  - their execution speed is **relatively slow**

Caffe + OpenCV

- **Network**
- **Mean**
- **Label**
- **Model**

4 files

Objective

- We create a ***Caffe2C*** 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

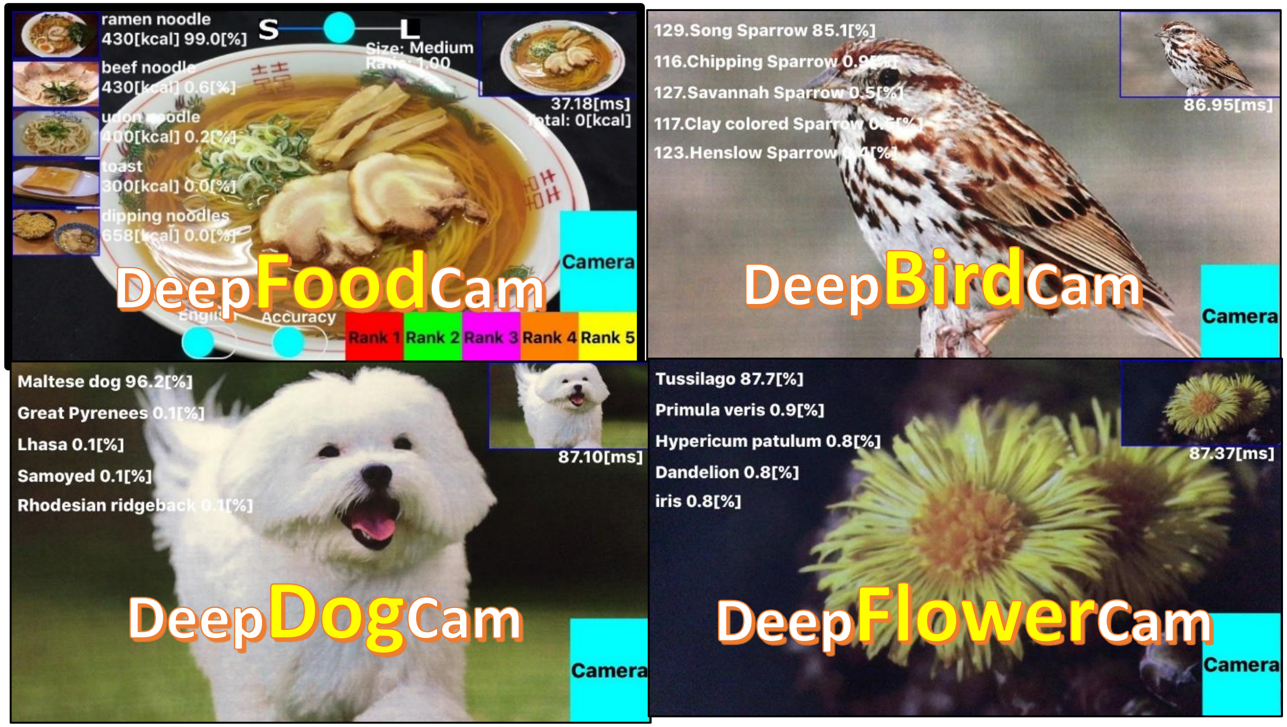
Caffe
- **Network**
- **Mean**
- **Label**
- **Model**

4 files $\xrightarrow{\textit{Caffe2C}}$ **Single C code**

- ***Caffe2C*** makes it easy to use deep learning on the C language operating environment
- ***Caffe2C*** achieves **faster runtime** in comparison to the existing OpenCV DNN module

Objective

- In order to demonstrate the utilization of the ***Caffe2C***, we have implemented 4 kinds of mobile CNN-based image recognition apps on iOS.



Contributions

1. We create a ***Caffe2C*** which converts the model definition files and the parameter files of Caffe into a single C code that can run on mobile devices

2. We explain the flow of construction of recognition app using ***Caffe2C***

3. We have implemented 4 kinds of mobile CNN-based image recognition apps on iOS.

2. CONSTRUCTION OF CNN-BASED MOBILE RECOGNITION SYSTEM

Caffe2C

- In order to use the learned parameters by Caffe on mobile devices, it is necessary to currently use the **OpenCV DNN module** ➡ **not optimized**, **relatively slow**

- We create a *Caffe2C* which **converts the CNN model definition files** and the **parameter files trained by Caffe to a single C language code**
  - We can use parameter files trained by Caffe on mobile devices

Caffe2C

- *Caffe2C* achieves faster execution speed in comparison to the existing OpenCV DNN module

Runtime[ms] Caffe2C vs. OpenCV DNN(Input size: 227x227)

	Caffe2C	**OpenCV DNN**
	AlexNet	
iPhone 7 Plus	**106.9**	1663.8
iPad Pro	**141.5**	1900.1
iPhone SE	**141.5**	2239.8

Speedup Rate: About 15X～

Reasons for Fast Execution

1. *Caffe2C* directly converts the Deep Neural Network to a C source code

Caffe

Network
Mean
Label
Model

Caffe2C

→ **Single C code**

Execution like **Compiler**

Execution like **Interpreter**

Reasons for Fast Execution

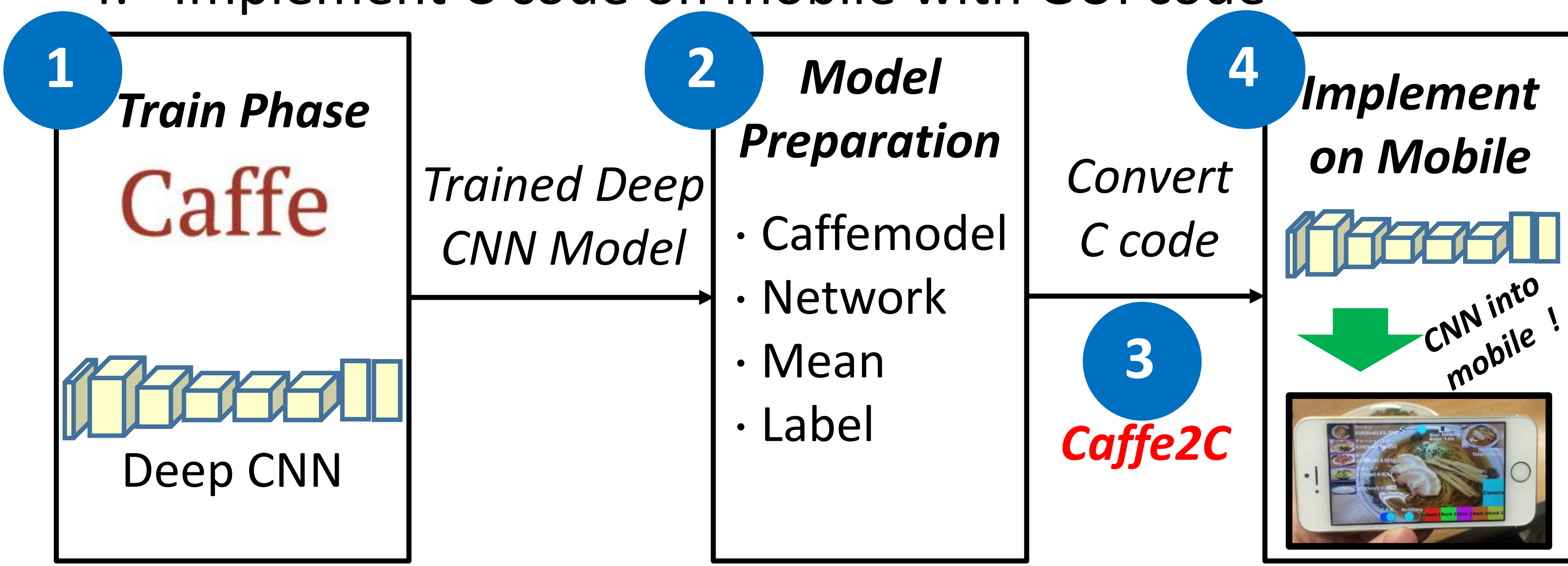
2. *Caffe2C* performs the pre-processing of the CNN as much as possible to reduce the amount of online computation
   – Compute batch normalization in advance for conv weight.

Caffe { • Network • Mean • Label • Model } 4 files $\xrightarrow{\textit{Caffe2C}}$ Single C code

3. *Caffe2C* effectively uses NEON/BLAS by multi-threading

Deployment Procedure

1. Train Deep CNN model by Caffe
2. Prepare model files
3. Generate a C source code by *Caffe2C* automatically
4. Implement C code on mobile with GUI code



3. IMAGE RECOGNITION SYSTEM FOR EVALUATION

Image Recognition System for evaluation

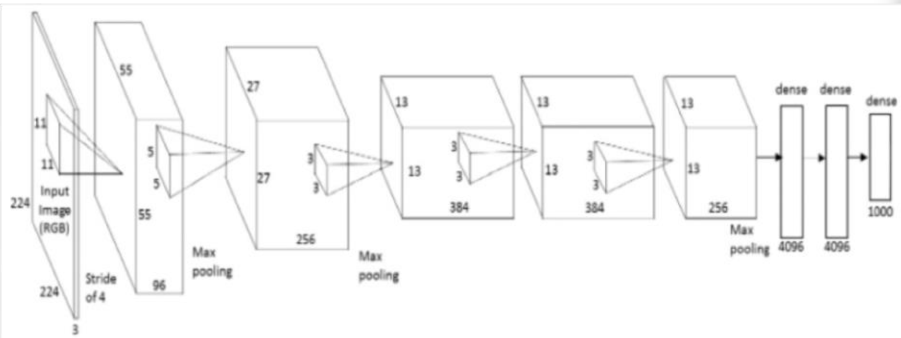
- In order to demonstrate the utilization of the Caffe2C, we have implemented four kinds of mobile CNN-based image recognition apps on iOS

- We explain image recognition engine used in the iOS application

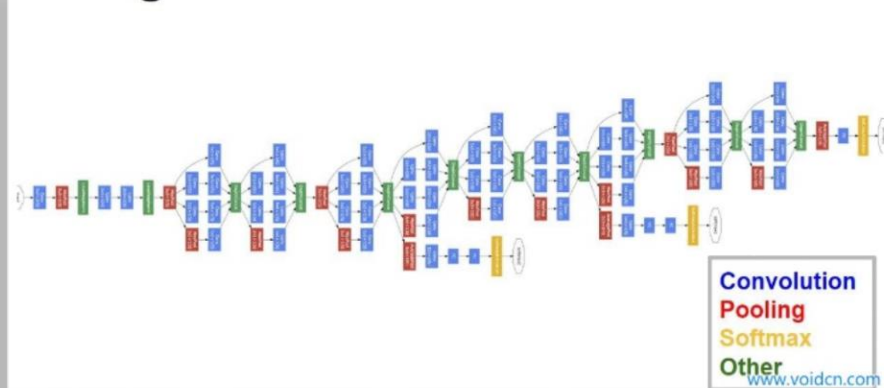
CNN Architecture

- A representative architectures are AlexNet VGG-16 GoogleNet or NIN

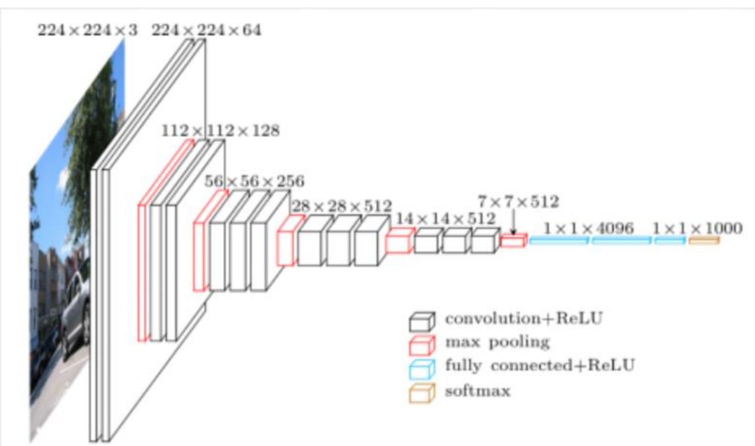
AlexNet



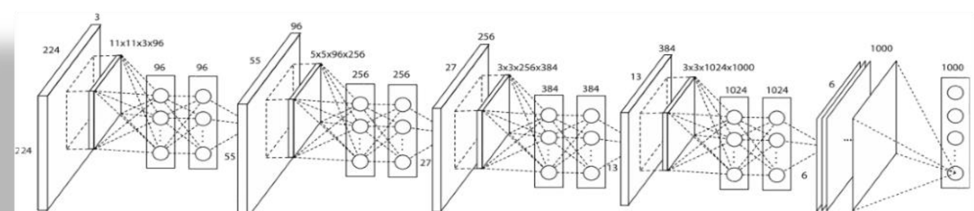
GoogLeNet



VGG-16



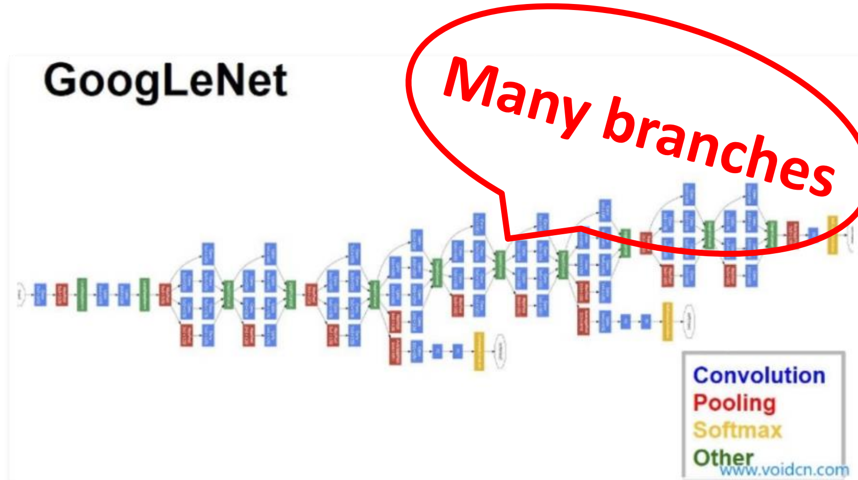
Network-In-Network



CNN Architecture

- The number of weights in AlexNet and VGG-16 is too much for mobile.

- GoogLeNet is too complicated for efficient parallel implemen-tation. (It has many branches.)



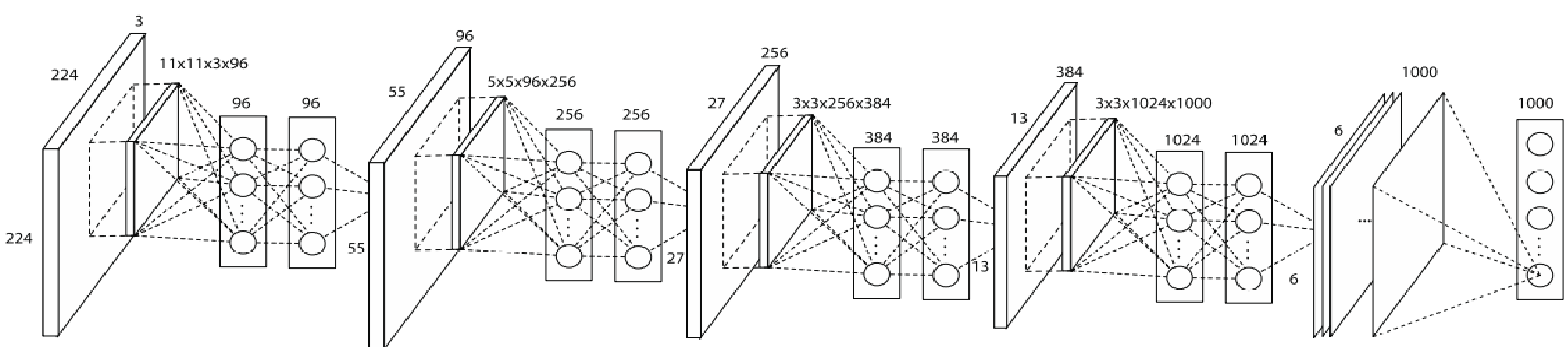
model		Alex	VGG-16	GoogLeNet	NIN
conv	layer	5	13	21	12
	weights	3.8M	15M	5.8M	7.6M
	comp.	1.1B	15.3B	1.5B	1.1B
FC	layer	3	3	1	0
	weights	59M	124M	1M	0
	comp.	59M	124M	1M	0
TOTAL	weights	62M	138M	6.8M	7.6M
	comp.	1.1B	15.5B	1.5B	1.1B
ImageNet	top-5 err.	17.0%	7.3%	7.9%	10.9%

CNN Architecture

- We adopt **Network-in-Network (NIN).**
  - No fully-connected layers (which bring less parameters)
  - Straight flow and consisting of many conv layers
  - relatively smaller than the other architectures

    ⇒ It's easy for parallel implementation.
    Efficient computation for conv layers is needed !

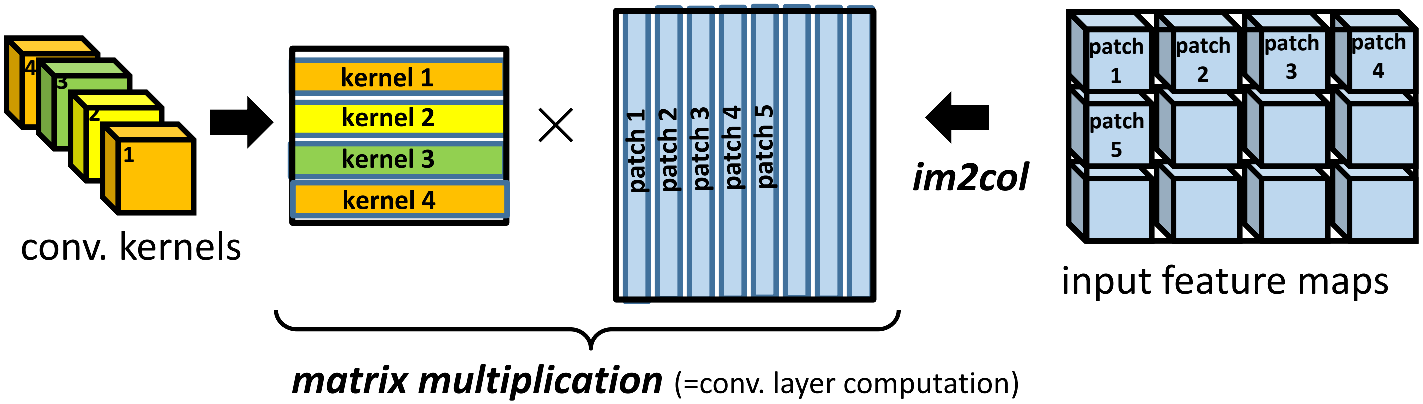
Network-In-Network(NIN)



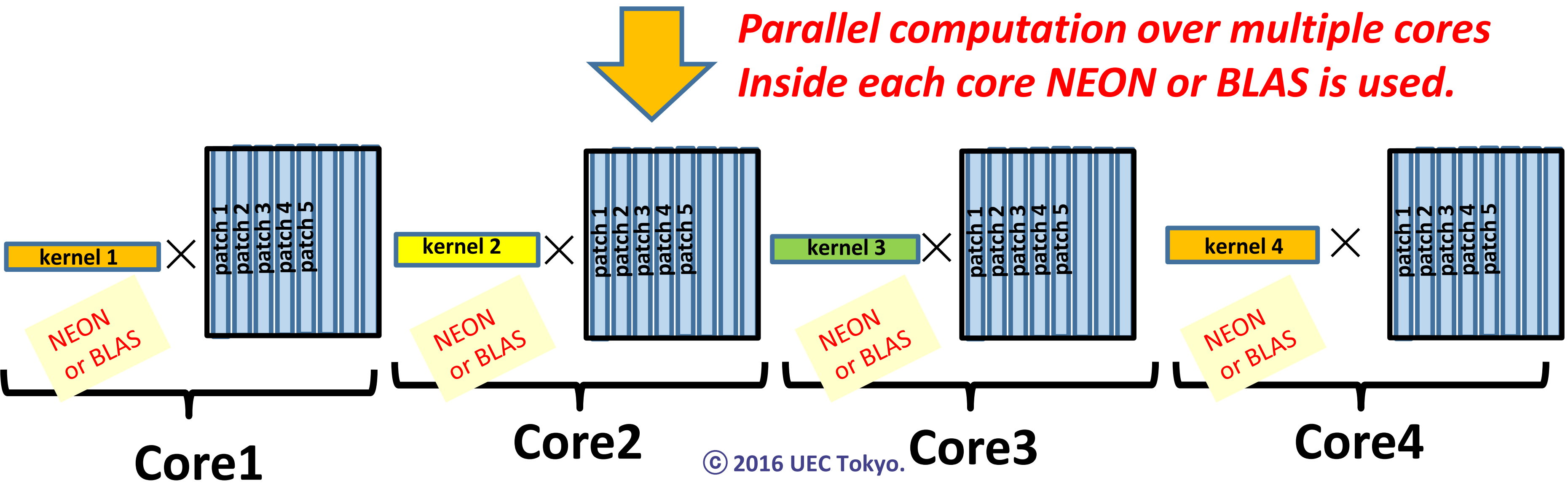
Fast computation of conv layers

- *efficient GEMM with 4 cores and BLAS/NEON* -

- Conv = im2col + GEMM (Generic Matrix Multiplication)



Parallel computation over multiple cores
Inside each core NEON or BLAS is used.



Fast Implementation on Mobile

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

Evaluation: Processing time

- **iOS**: **BLAS >> NEON**, **Android**: **BLAS << NEON**
  - For iOS, using BLAS in the iOS Accelerate Framework is the best choice.
  - For Android, using NEON (SIMD instruction) is better than OpenBLAS.

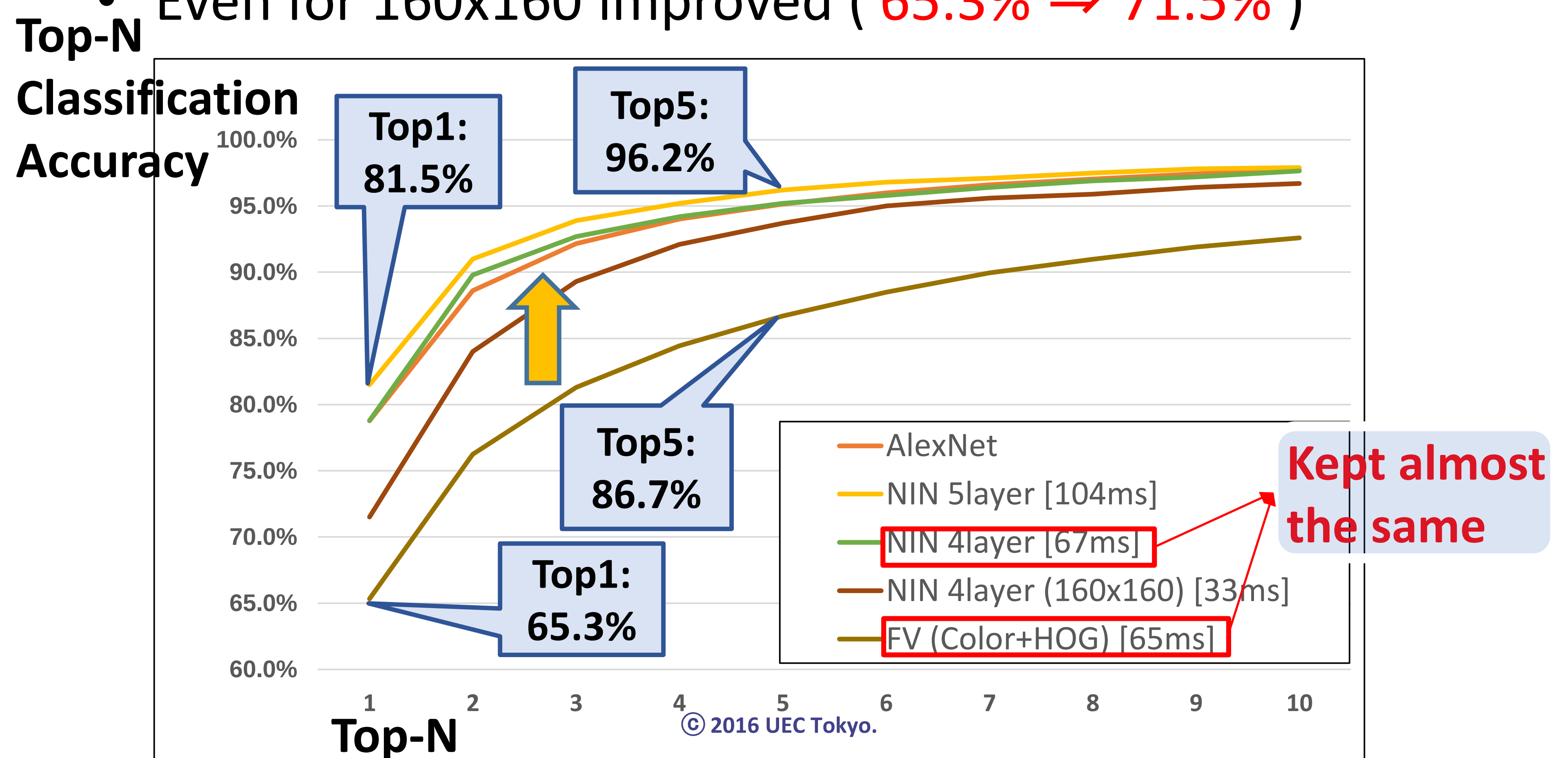
Recognition Time[ms] BLAS vs. NEON

	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	**251.0**	1652.0	GALAXY Note 3	OpenBLAS

Highly optimized

Comparison to FV-based Previous Method Deep Learning with UEC-FOOD100 dataset

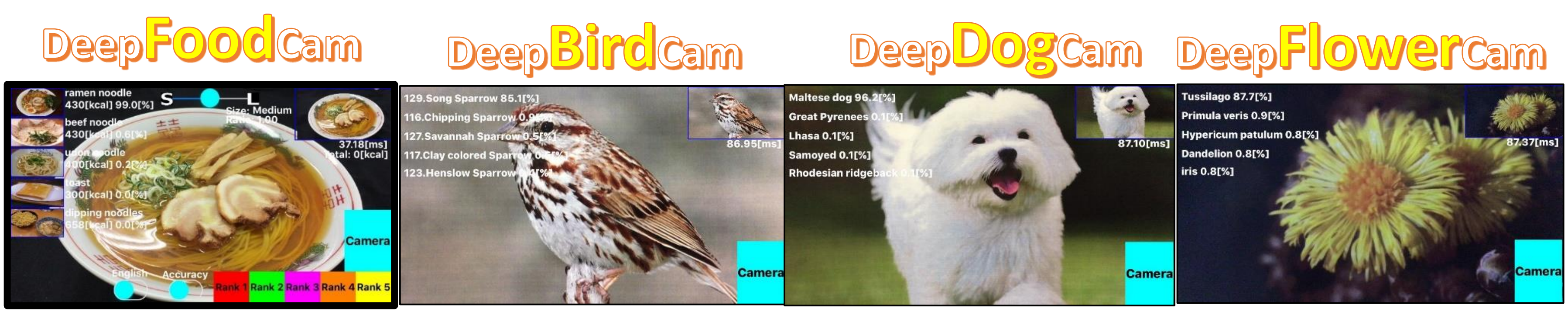
- Much improved (65.3% ⇒ 81.5% (top-1))
- Even for 160x160 improved (65.3% ⇒ 71.5%)



4. MOBILE APPLICATIONS

4 iOS Applications

- We have implemented 4 kinds of mobile CNN-based image recognition apps on iOS
  - Food recognition app: "*DeepFoodCam*"
  - Bird recognition app: "*DeepBirdCam*"
  - Dog recognition app: "*DeepDogCam*"
  - Flower recognition app: "*DeepFlowerCam*"



DeepFoodCam

- Recognize 101 classes including 100 food classes and one nonfood class

Training Phase

- fine-tuned the CNN with 101 class images
  - totally 20,000 images
  - UECFOOD-100 and non-food collected from Twitter

Accuracy

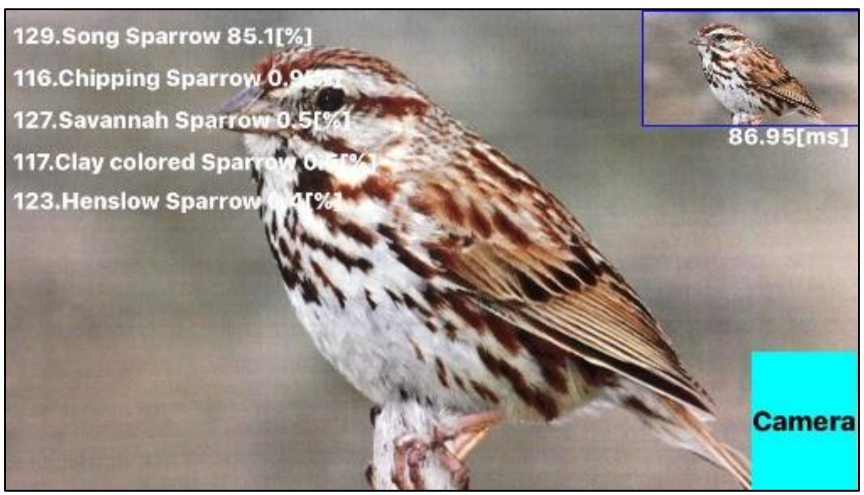
Target	Top-1	Top-5
Food 101 class	74.5%	93.5%

DeepBirdCam

- Recognize 200 bird class

Training Phase

- fine-tuning CNN with 6033 images of Caltech-UCSD Birds 200 Dataset



Accuracy

Target	Top-1	Top-5
Bird 200 class	55.8%	80.2%

DeepDogCam

- Recognize 100 dog class

Training Phase

- fine-tuning CNN with 150 and over images per class of Stanford Dogs Dataset Dataset

Accuracy

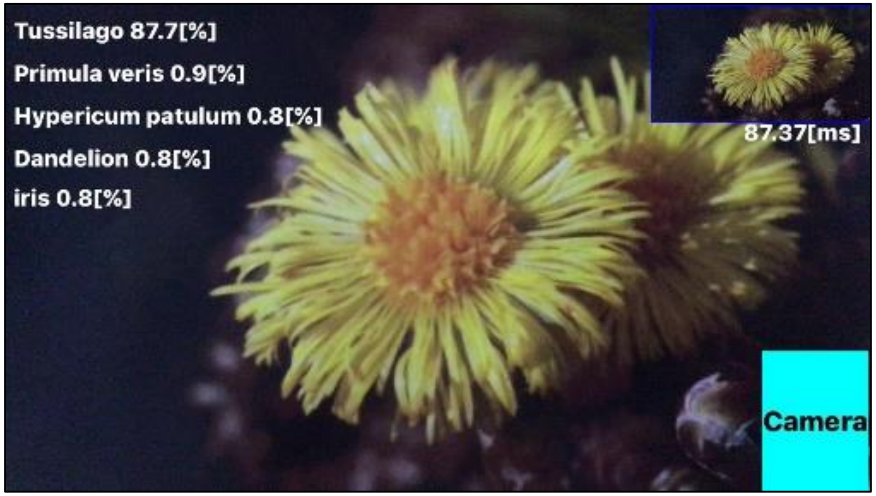
Target	Top-1	Top-5
Dog 100 class	69.0%	91.6%

DeepFlowerCam

- Recognize 102 flower class

Training Phase

- fine-tuning CNN with 80 and over images per class of 102 Category Flower Dataset



Accuracy

Target	Top-1	Top-5
Flower 102 class	64.1%	85.8%

4 iOS Applications

- We have implemented 4 kinds of mobile CNN-based image recognition apps on iOS
  - Food recognition app: *“DeepFoodCam”*
  - Bird recognition app: *“DeepBirdCam”*
  - Dog recognition app: *“DeepDogCam”*
  - Flower recognition app: *“DeepFlowerCam”*

If you prepare training data, you can create mobile recognition apps in a day !!

DeepFoodCam

DeepBirdCam

DeepDogCam

DeepFlowerCam

Conclusions

1. We create a ***Caffe2C*** which converts the model definition files and the parameter files of Caffe into a single C code that can run on mobile devices

2. We explain the flow of construction of recognition app using ***Caffe2C***

3. We have implemented 4 kinds of mobile CNN-based image recognition apps on iOS.

Additional work

- We implemented apply our mobile framework into real-time CNN-based mobile image processing
  - such as **Neural Style Transfer**

iOS App is Available !

“**DeepFoodCam**“

Object Recognition

Thank you for listening

Neural Style Transfer

iOS App is Available !

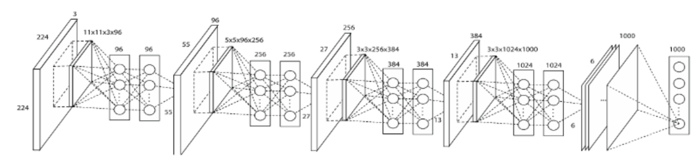
“**RealTimeMultiStyleTransfer**”

The University of Electro-Communications

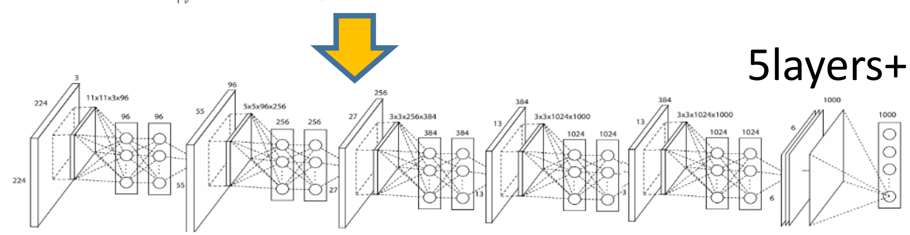
Extension of NIN

adding BN, 5layers, multiple image size

- Modified models (BN, 5layer, multi-scale)
  - adding BN 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**

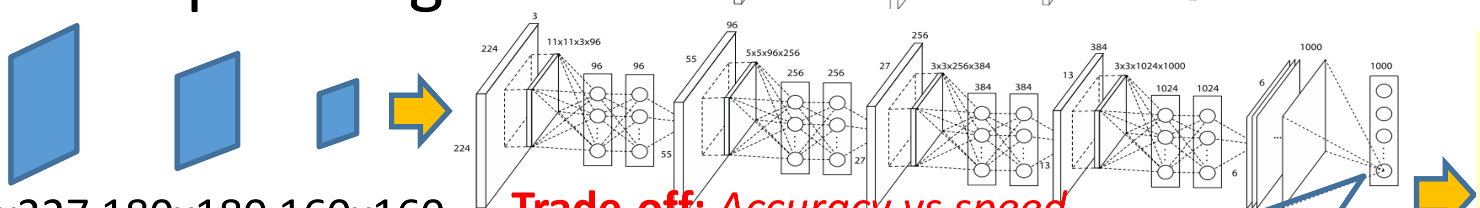


4layers



5layers+BN

- Multiple image size



227x227 180x180 160x160

Trade-off: *Accuracy vs speed*

Global Average Pooling (GAP)

227x227
55.7ms 78.8%
180x180
35.5ms 76.0%
160x160
26.3ms 71.5%