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 Computer 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

© 2016 UEC Tokyo.
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

© 2016 UEC Tokyo.
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
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.

  ![Diagram](image)

- **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.

DeepFoodCam
- Maltese dog 96.2%
- Great Pyrenees 0.1%
- Lhasa 0.1%
- Samoyed 0.1%
- Rhodesian ridgeback 0.1%

DeepBirdCam
- Dussilago 87.7%
- Primula veris 0.9%
- Hypericum patulum 0.8%
- Dandelion 0.8%
- Iris 0.8%

DeepDogCam
- Ramen noodle 430(kcal) 99.0%
- Beef noodle 430(kcal) 99.0%
- Pork noodle 180(kcal) 92.1%
- Seafood 250(kcal) 99.9%
- Beef and noodle 650(kcal) 0.0%

DeepFlowerCam
- Song Sparrow 85.1%
- Chipping Sparrow 0.9%
- Savannah Sparrow 9.7%
- Clay colored Sparrow 0.6%
- Henslow Sparrow 0.1%

© 2016 UEC Tokyo.
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)

<table>
<thead>
<tr>
<th>Device</th>
<th>Caffe2C</th>
<th>OpenCV DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone 7 Plus</td>
<td>106.9</td>
<td>1663.8</td>
</tr>
<tr>
<td>iPad Pro</td>
<td>141.5</td>
<td>1900.1</td>
</tr>
<tr>
<td>iPhone SE</td>
<td>141.5</td>
<td>2239.8</td>
</tr>
</tbody>
</table>

Speedup Rate: About 15X~
Reasons for Fast Execution

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

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

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 implementation. (It has many branches.)

<table>
<thead>
<tr>
<th>model</th>
<th>Alex</th>
<th>VGG-16</th>
<th>GoogLeNet</th>
<th>NIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>layer</td>
<td>5</td>
<td>13</td>
<td>21</td>
<td>12</td>
</tr>
<tr>
<td>weights</td>
<td>3.8M</td>
<td>15M</td>
<td>5.8M</td>
<td>7.6M</td>
</tr>
<tr>
<td>comp.</td>
<td>1.1B</td>
<td>15.3B</td>
<td>1.5B</td>
<td>1.1B</td>
</tr>
<tr>
<td>FC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>layer</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>weights</td>
<td>59M</td>
<td>124M</td>
<td>1M</td>
<td>0</td>
</tr>
<tr>
<td>comp.</td>
<td>59M</td>
<td>124M</td>
<td>1M</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weights</td>
<td>62M</td>
<td>138M</td>
<td>6.8M</td>
<td>7.6M</td>
</tr>
<tr>
<td>comp.</td>
<td>1.1B</td>
<td>15.5B</td>
<td>1.5B</td>
<td>1.1B</td>
</tr>
<tr>
<td>ImageNet</td>
<td>top-5 err.</td>
<td>17.0%</td>
<td>7.3%</td>
<td>7.9%</td>
</tr>
</tbody>
</table>
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 $\rightarrow$ Speeding up GEMM
  – computation of conv layer 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)
    • Total: iOS: 2Core $\times$ 4 = 8 calculation, Android: 4Core $\times$ 4 = 16 calculation

• **BLAS library** (highly optimized for iOS $\Leftrightarrow$ 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.

<table>
<thead>
<tr>
<th>Devices</th>
<th>NEON</th>
<th>BLAS</th>
<th>BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>181.0</td>
<td>55.7</td>
<td>Accelerate</td>
</tr>
<tr>
<td>iOS</td>
<td>222.4</td>
<td>66.0</td>
<td>Accelerate</td>
</tr>
<tr>
<td>iOS</td>
<td>251.8</td>
<td>79.9</td>
<td>Accelerate</td>
</tr>
<tr>
<td>Android</td>
<td>251.0</td>
<td>1652.0</td>
<td>OpenBLAS</td>
</tr>
</tbody>
</table>
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%)

Top-N Classification Accuracy

- Top1: 81.5%
- Top5: 96.2%
- Top1: 86.7%
- Top5: Kept almost the same

- AlexNet
- NIN 5layer [104ms]
- NIN 4layer [67ms]
- NIN 4layer (160x160) [33ms]
- FV (Color+HOG) [65ms]
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

<table>
<thead>
<tr>
<th>Target</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Food 101 class</td>
<td>74.5%</td>
<td>93.5%</td>
</tr>
</tbody>
</table>
DeepBirdCam

- Recognize 200 bird class

Training Phase

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

<table>
<thead>
<tr>
<th>Target</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bird 200 class</td>
<td>55.8%</td>
<td>80.2%</td>
</tr>
</tbody>
</table>
DeepDogCam

• Recognize 100 dog class

Training Phase

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

Accuracy

<table>
<thead>
<tr>
<th>Target</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog 100 class</td>
<td>69.0%</td>
<td>91.6%</td>
</tr>
</tbody>
</table>

Maltese dog 96.2[%]
Great Pyrenees 0.1[%]
Lhasa 0.1[%]
Samoyed 0.1[%]
Rhodesian ridgeback 0.1[%]
DeepFlowerCam

• Recognize 102 flower class

Training Phase

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

Accuracy

<table>
<thead>
<tr>
<th>Target</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flower 102 class</td>
<td>64.1%</td>
<td>85.8%</td>
</tr>
</tbody>
</table>
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 !!
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
Thank you for listening

Object Recognition

iOS App is Available!
“DeepFoodCam”

iOS App is Available!
“RealTimeMultiStyleTransfer”
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

• Multiple image size

  227x227 180x180 160x160

Trade-off: Accuracy vs speed

Global Average Pooling (GAP)