CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android

Salar Latifi, Hossein Golestani, Matin Hashemi, Soheil Ghiasi

Sharif University of Technology

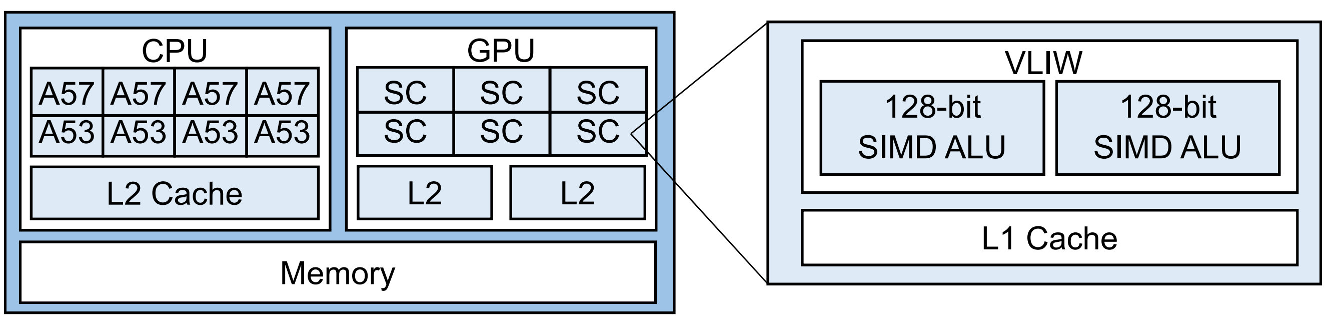
University of California, Davis

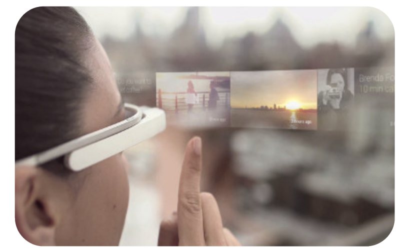
Source Code and Example Apps: github.com/ENCP/CNNdroid

Contact E-mail: matin@sharif.edu

CNNdroid Motivation

- GPU acceleration enables practical execution of many deep CNN algorithms on smartphones and wearable devices and enables more creative media-rich apps
- CNNdroid is the first GPU-accelerated library for execution of trained deep CNNs on Android devices
  - Android versions of Caffe and Torch only employ multi-core mobile CPU and not mobile GPU
- Mobile GPU architecture is different from desktop GPU, hence, it is impossible / inefficient to port existing parallel algorithms in desktop libraries, e.g., Theano, to Android





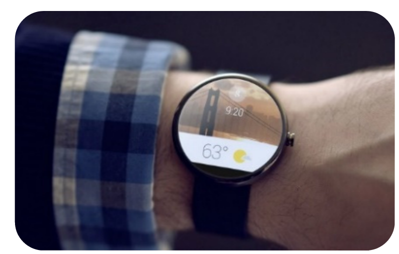
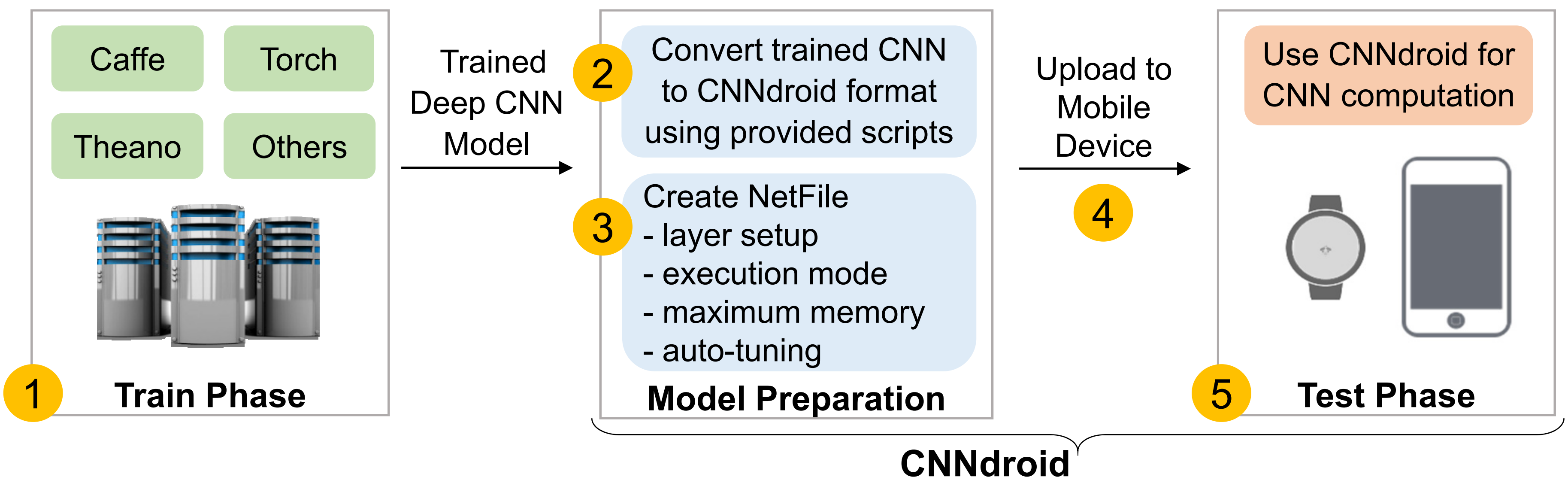


Image Credits: IBM Research, Guardianlv, Nixie, Android Wear

Deployment Procedure

1. CNN model is trained by desktop platforms, e.g., Caffe, Torch, Theano
2. CNNdroid scripts convert trained models into CNNdroid format
3. NetFile is created by the user
    - Layer setup of the deep CNN model
    - Execution mode: CPU-only or GPU-accelerated
    - Maximum memory usage of the library
    - Turn auto-tuning ON or OFF
4. The trained model and the NetFile are uploaded to mobile device
5. Integration of CNNdroid into target Android app in only 5 lines of code



Integration into Android application

```
// 1) Import CNNdroid library import network.CNNdroid;
 ...

// 2) Construct Renderscript object
RenderScript myRenderScript = RenderScript.create(this);

// 3) Provide NetFile and construct CNNdroid object
String NetFile = "/sdcard/AlexNet/AlexNet_NetFile.txt";
CNNdroid myCNN = new CNNdroid(myRenderScript, NetFile);

// 4) Prepare your input, which can be
//    a single image or a batch of images
float[][][]  inputSingle = loadSingleInput();
float[][][][] inputBatch = loadBatchInput();

// 5) Call 'compute' function for CNN execution
//    and receive the result as an object
Object output = myCNN.compute(inputBatch);
```

Experiment Results - Speedup

- Average runtime of the heaviest convolution layer per image in a batch of 16 images, and the corresponding speedup
- Up to ~ 60 X speedup

		Sequential Runtime (ms)	Accelerated Runtime (ms)	Speedup Rate
Samsung Galaxy Note 4	LeNet-5	44	1.8	24.44
	Alex’s CIFAR-10	162	7.5	21.6
	AlexNet	5876	92.6	63.45
HTC One M9	LeNet-5	62	4.3	14.42
	Alex’s CIFAR-10	168	8.7	19.31
	AlexNet	5828	152	38.34

Experiment Results - Speedup

- Average runtime of the entire CNN execution per image in a batch of 16 images, and the corresponding speedup
- Up to ~ 40 X speedup

		Sequential Runtime (ms)	Accelerated Runtime (ms)	Speedup Rate
Samsung Galaxy Note 4	LeNet-5	62	12.8	4.84
	Alex's CIFAR-10	313	25.3	12.37
	AlexNet	20767	481.7	43.11
HTC One M9	LeNet-5	81	16.6	4.88
	Alex's CIFAR-10	326	31	10.51
	AlexNet	21382	709	30.16

Experiment Results - Energy

- Power & energy consumption per image for AlexNet on HTC One M9
- Measured by “Qualcomm Trepn Profiler” application
- Average of multiple measurements with ~ 20 % variability
- ~ 130 X saving in energy

	Sequential	Accelerated	Reduction
Power (mW)	2337.70	522.87	~ 4.5 X
Energy (J)	51.6	0.4	~ 130 X