Deep Learning: Low Power and Parallel Computation On Android Phone

EE202B by Prof. Mani Srivastava, Final Project

Participants: Zhengshuang Ren, Yingnan Wang
Acknowledgment: Moustafa Alzantot

Date: March 22, 2017

Technology Keywords

- **TensorFlow**
    - Eigen: the default BLAS library used in TensorFlow (32-bit floating point)
    - GEMMLowp: eight-bit-int general matrix multiplication library
- **RenderScript**: a framework for running computationally intensive tasks at high performance on Android
- **Android NDK**: Native Development Kit is a tool that allows you to program in C/C++ for Android devices.
- **Bazel**: Google's own build tool, the only official build tool fully supported by TensorFlow
- **Trepn Profiler**: Useful app to monitor the Android system performance.

Introduction

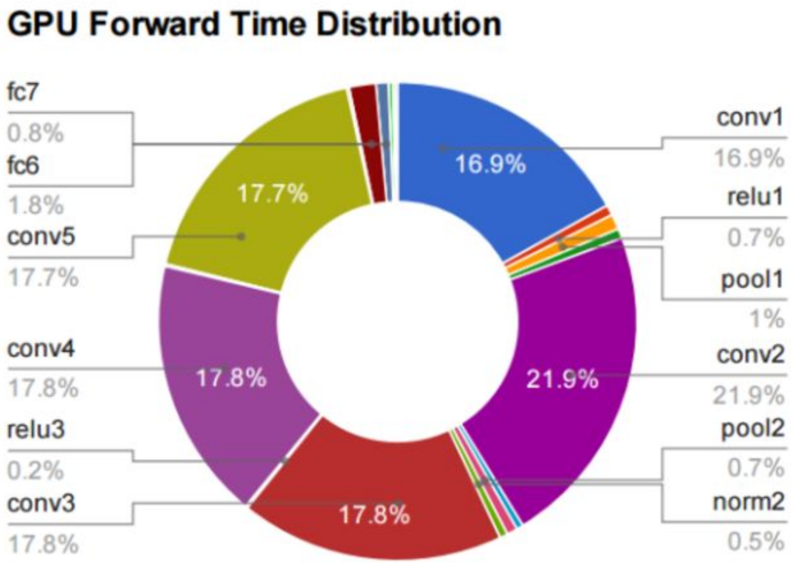
- Deep Learning technique shifts to lower power devices
- Android Phone evolves with more powerful hardware
- Mobile device is limited by power constraint
- TensorFlow does not have good support for low power usage
- Goal: Improve speed and power consumption of Android App using TensorFlow in three levels: Data level, Op level and Android level

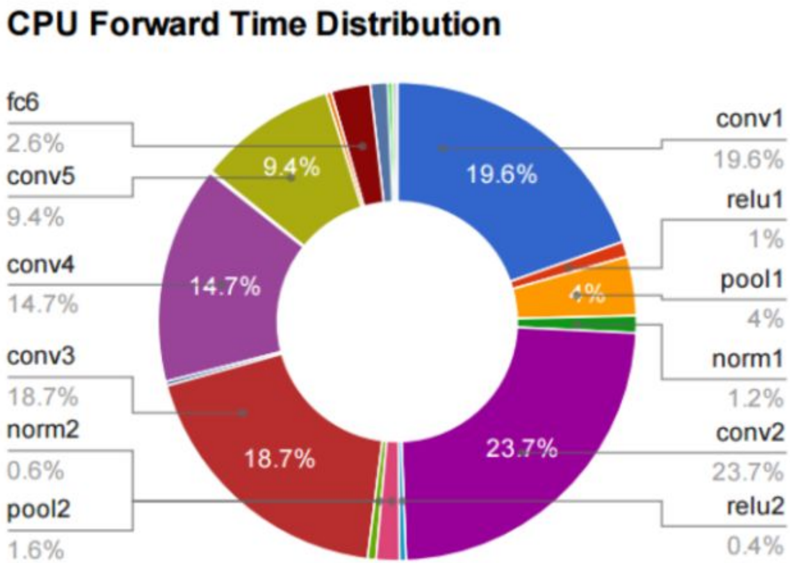
Previous Work

- SqeezeNet:
    - Fast and Energy-Efficient CNN Inference on IoT Devices by Mohammad et al
    - Implemented a specific model on the Android phone with RenderScript
    - Paper: https://arxiv.org/pdf/1611.07151.pdf
    - Github: https://github.com/mtmd/Mobile_ConvNet

Background

Time Distribution for a typical neural network





Matmul and Conv takes up to 95% of the GPU computation time, and 89% on CPU!

Eigen vs. GEMMLowp

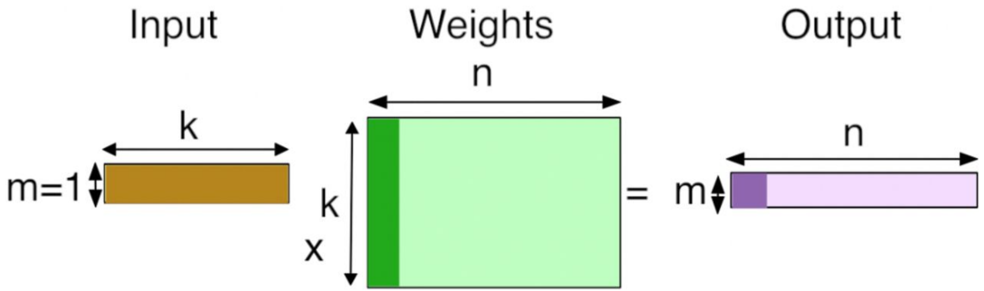
- **Eigen**:
  - Multithreading, 32-bit floating point
- **GEneral Matrix to Matrix Multiplication Low Precision**
  - Data quantization: from 32-bit to 8-bit
  - Tradeoff between accuracy and power/speed

GEMMLowp - Why?

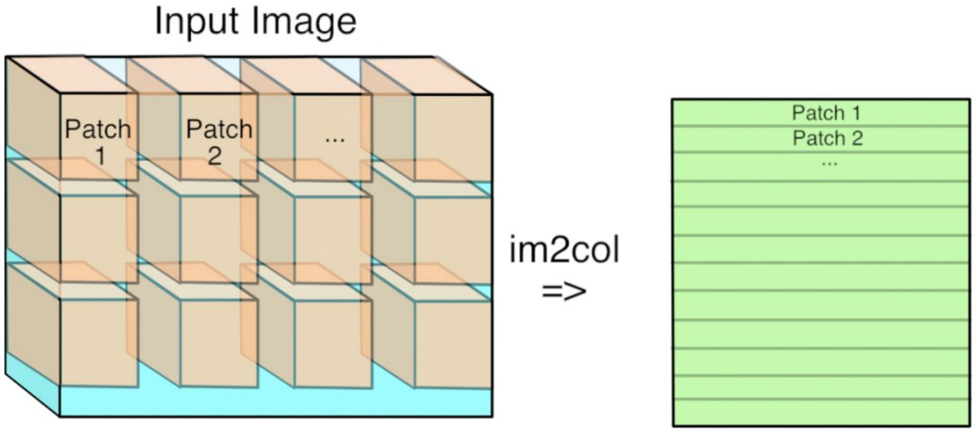
- Why does it work?
  - High precision input contains: noise, lighting change and other non-essential information
  - Non-essential information has much less weight than features (remain in lowp input)
- Why quantize?
  - Speed
  - Memory
  - Power Consumption

GEMMLowp - How?

Fully-Connected Layer:

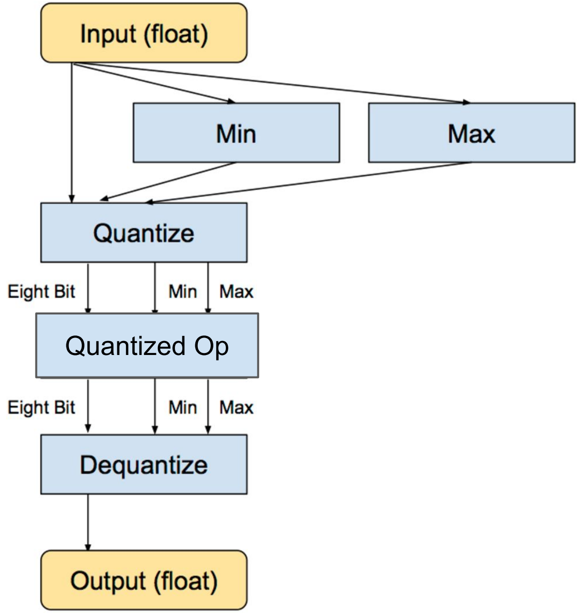


Convolutional Layer:

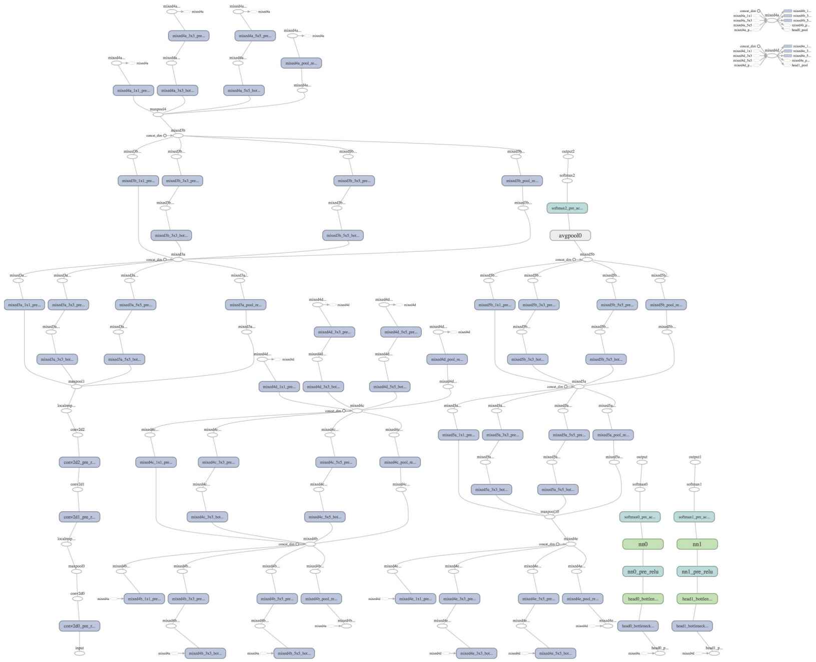


GEMMLowp - With TF

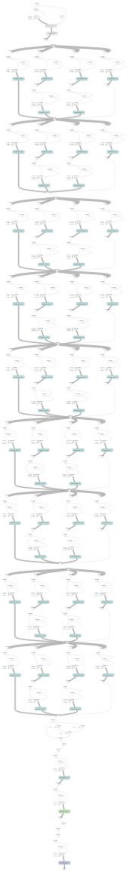
- Quantize the GraphDef protobuf file
- Same float input/output with internally 8-bit int operations
  - Ops
  - Input/Output Data
  - Weights



Inception of different version



Eigen

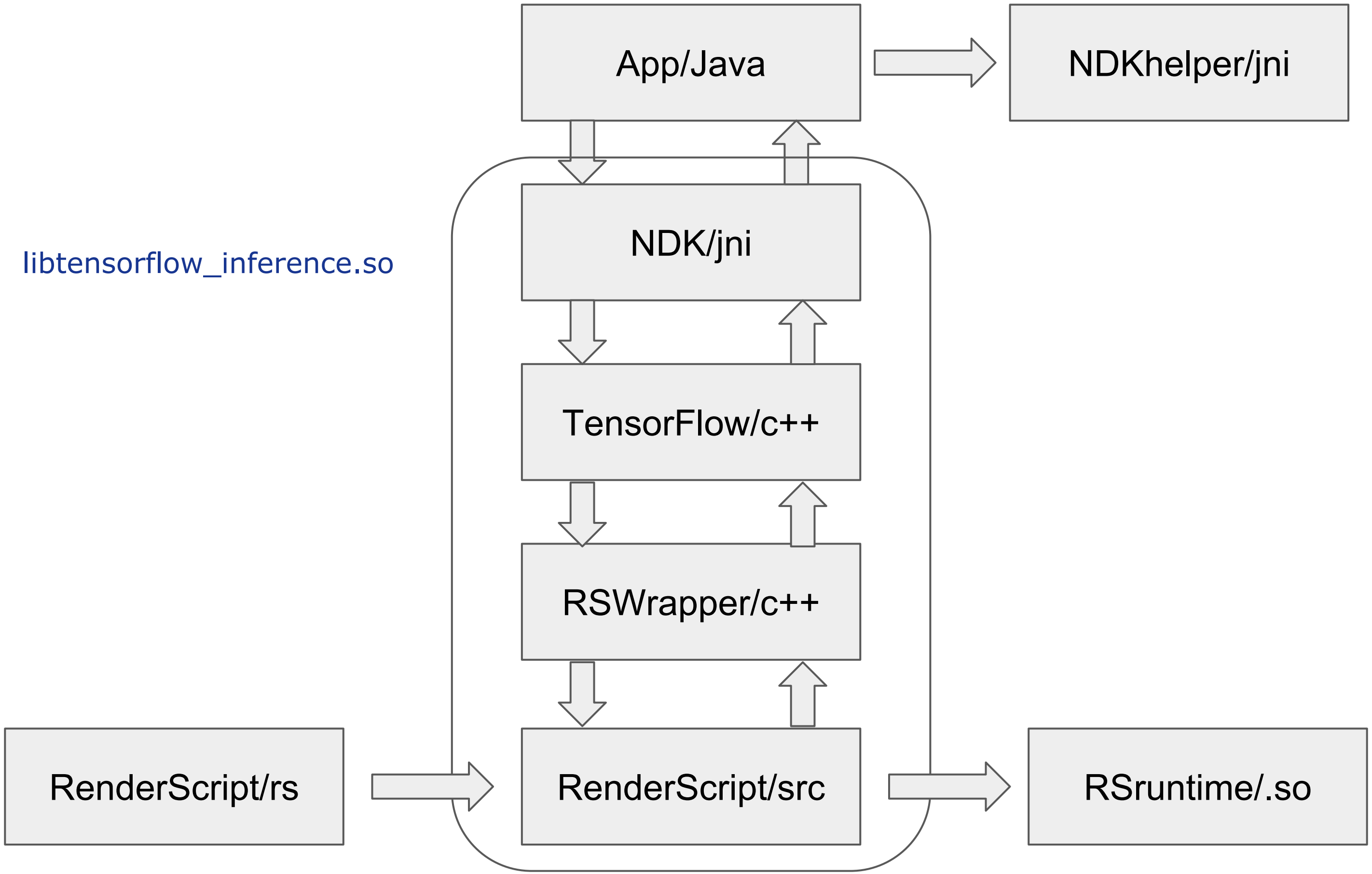


gemmlowp

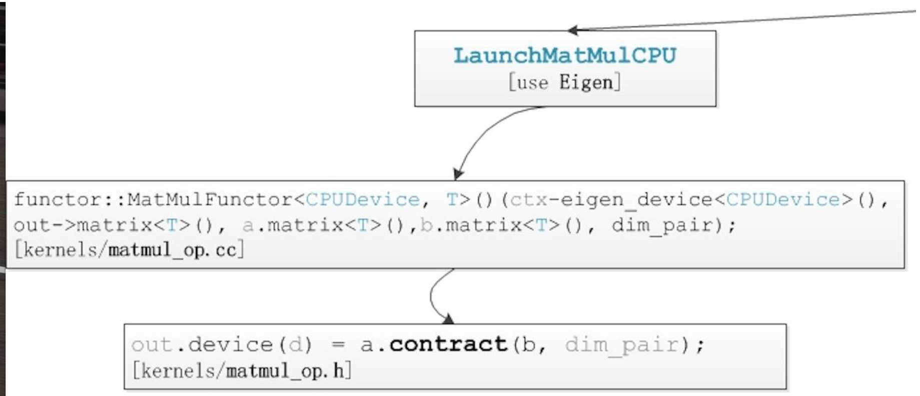
RenderScript Basic

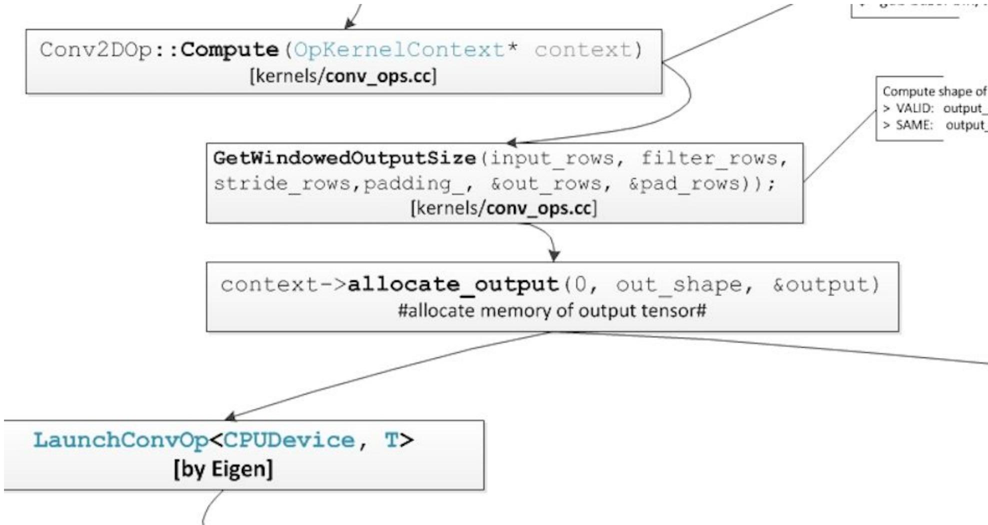
- C99, like CUDA
- Parallelly running on multi-core CPUs and GPUs
- Asynchronous execution
- Key components:
  - Invokable functions
  - Script globals
  - Compute kernels
- Reflect compiler: llvm-rs-cc
  - Header
  - Source file
  - Shared library
  - Object file
- Already has some built-in intrinsic kernels

Overall App Architecture

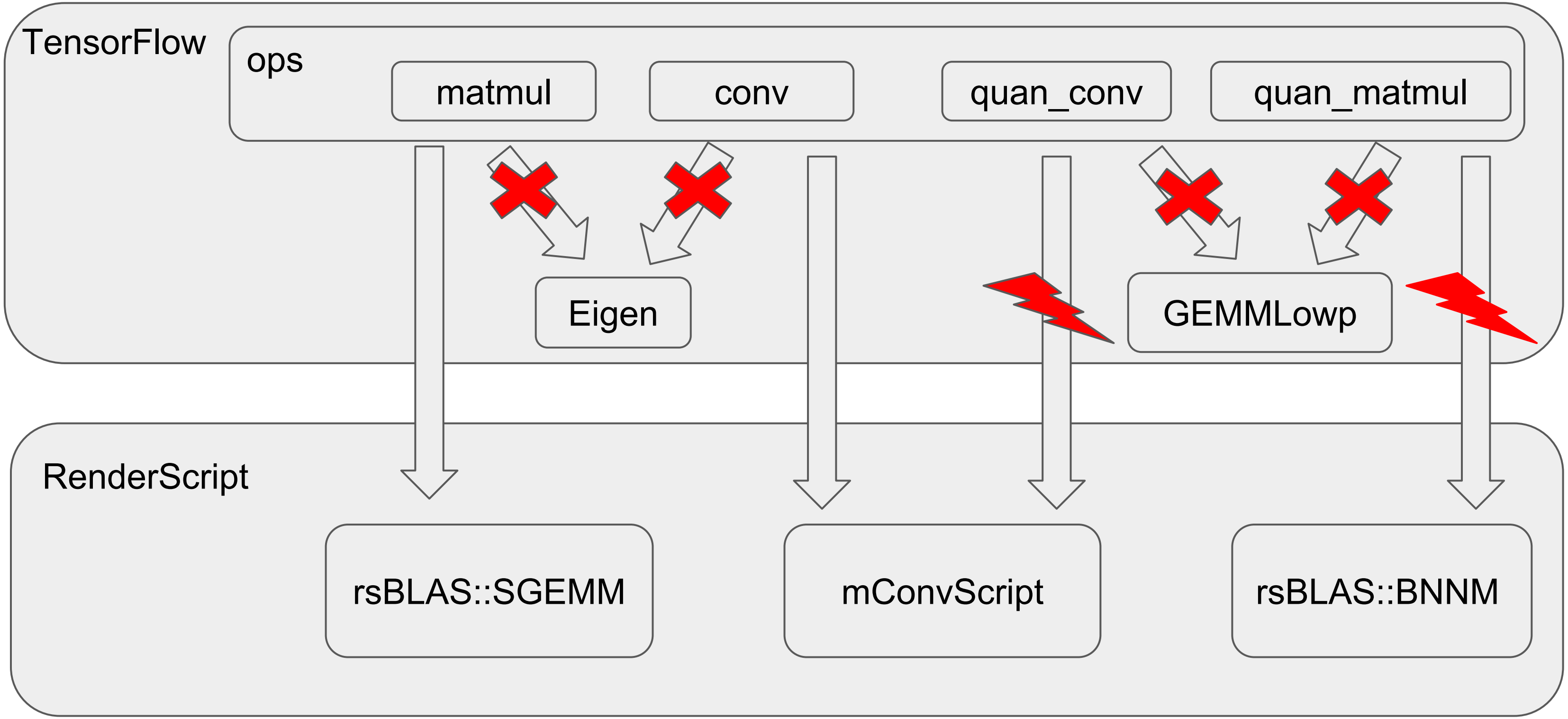


Hijack the ops





What we have done



Renderscript runtime API

```
// float
void rsMatmul_sgemm(void* a_ptr, bool a_trans, void* b_ptr, bool b_trans, void* c_ptr,
                    int m, int n, int k, float alpha, float beta);
// uint8_t
void rsMatmul_bnnm(void* a_ptr, int a_off, void* b_ptr, int b_off, void* c_ptr, int c_off,
                    int m, int n, int k, int c_mult);
// only 3x3 and 5x5 filter
template <typename T>
void rsConv_intrinsic(void* filter, void* input, void* output, rsConvInfo convInfo);
// custom setting
template <typename T>
void rsConv_script(void* filter, void* input, void* output, rsConvInfo convInfo);
```

RenderScript Op test result

```
D/NDK_LOG: +-+-+-+-Matmul Float test-+-+-+-+
V/RenderScript: Successfully loaded runtime: libRSDriver_adreno.so
I/NDK_LOG: rsMatmul small test passed!
I/NDK_LOG: rsMatmul medium test passed!
I/NDK_LOG: rsMatmul large test passed!
I/NDK_LOG: rsMatmul float TF test passed!
D/NDK_LOG: +-+-+-+-Matmul uint8_t test-+-+-+-+
I/NDK_LOG: rsMatmul small test passed!
I/NDK_LOG: rsMatmul medium test passed!
I/NDK_LOG: rsMatmul large test passed!
D/NDK_LOG: +-+-+-+-Conv Float test-+-+-+-+
I/NDK_LOG: rsConv_intrinsic 3x3 float small test passed!
I/NDK_LOG: rsConv_script 1x1 float TF test passed!
I/NDK_LOG: rsConv_script 3x3 float TF test passed!
I/NDK_LOG: rsConv_script 5x5 float TF test passed!
I/NDK_LOG: rsConv_script 7x7 float TF test passed!
D/NDK_LOG: +-+-+-+-Conv uint8_t test-+-+-+-+
I/NDK_LOG: TODO in the future
D/NDK_LOG: Time taken: 2.20921900s
```

Implementation details

- Static RS context so that only load the GPU runtime library once.
- Allocate memory at first time then reuse them
- BNNM use 1.10.21 fixed-point format. Need to shift left to maintain small value.
- TF data memory columnwise, RS Allocation rowwise
  - Out = conv(T(A), T(B)) = T(conv(A, B))
  - Intrinsic custom script
- Launch parallel computing kernels based on output elements
- Fix weird padding issue: 7x7 filter with padding of 2

Function Call of hijacking TensorFlow

Matmul:

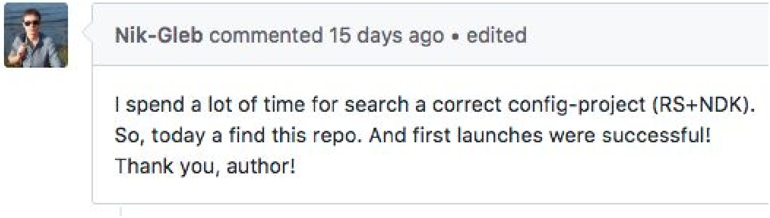
```
androidrs::matmul::rsMatmul_sgemm(static_cast<void*>(const_cast<char*>(a.tensor_data().data())), 0,
                                  static_cast<void*>(const_cast<char*>(b.tensor_data().data())), 0,
                                  static_cast<void*>(const_cast<char*>(out->tensor_data().data())),
                                  a.dim_size(0), b.dim_size(1), a.dim_size(1), 1, 0);
return;

LaunchMatMul<Device, T, USE_CUBLAS>::launch(ctx, this, a, b, dim_pair, out);
```

Conv:

```
androidrs::conv::rsConvInfo convInfo(in_depth, input_rows, input_cols, filter_rows, filter_cols,
                                     stride_rows, stride_cols, pad_rows, pad_cols,
                                     out_depth, out_rows, out_cols, batch, sizeof(T));
androidrs::conv::rsConv_script<T>(static_cast<void*>(const_cast<char*>(filter.tensor_data().data())),
                                  static_cast<void*>(const_cast<char*>(input.tensor_data().data())),
                                  static_cast<void*>(const_cast<char*>(output->tensor_data().data())),
                                  convInfo);
return;
launcher_.launch(context, use_cudnn_, cudnn_use_autotune_, input, filter,
                 stride_rows, stride_cols,
                 BrainPadding2EigenPadding(padding_), output, data_format_);
```

Build Procedure

- RS NDK support only has been added back to r14 since this March!
- RS NDK project currently can **only** be built with ndk-build, but TensorFlow **must** be built with Bazel. However, Bazel's NDK toolchain is still r12.
- Our Solution
  - Build RS with ndk-build and link the reflected source files with TF using bazel.
- The first on Internet to implement a RS NDK example



- Struggle a month with Bazel link ndk-built until today 3AM!

Issue

- Memory reuse
  - The hijack op method cannot communicate with the TF context so we cannot have a space of dynamic memory to reuse
  - This have huge memory overhead issue (~2sec)
- Solution
  - Don’t use dynamic container (vector)
  - Find a TF class has life time as the TF context and declare static object in it
  - Write a user op which is directly managed by the TF context

Reference Links

- Is it possible to call RenderScript in NDK with prebuilt libraries?
  - https://github.com/android-ndk/ndk/issues/304
- Android - Link TensorFlow with ndk-build prebuilt library
  - https://groups.google.com/forum/#!msg/bazel-discuss/N2OPrkXskqk/GALuknW5AQAJ
- How to hijack a TensorFlow op
  - http://stackoverflow.com/questions/42809657/how-to-construct-a-tensorflowtensor-from-raw-pointer-data-in-c/42815376?noredirect=1#comment72752184_42815376
- RenderScript NDK weird bugs
  - https://github.com/android-ndk/ndk/issues/331
- Bazel android example still need add --spawn_strategy=standalone when build on mac
  - https://github.com/bazelbuild/bazel/issues/2597

Experiment Setup

- Hardware: Nexus 5
  - CPU: Quad-core 2.3 GHz Snapdragon 800 processor
  - GPU: Adreno 330, 450MHz
  - Memory: 2 GB of LPDDR3-1600 RAM
- Software:
  - OS: Android 6.0.1
  - TensorFlow Android Demo App: Classifier
  - Inception v1 Model
  - Trepn Profiler

Procedure

- Make sure before each trial:
  - The only things running on the device is TF Classifier and Trepn Profiler besides OS
  - Similar Battery Remaining Level
  - Similar Battery Temperature & Processor Temperature
  - Unplugged
- Collect Baseline Energy Info Before Running TF Classifier
- Run TF Classifier
  - 3 times (5min each) and record data (16 data points)
  - 30 min, 8 data points
  - Profiling at a 100 ms interval
- Analyze Data using python script

Evaluation

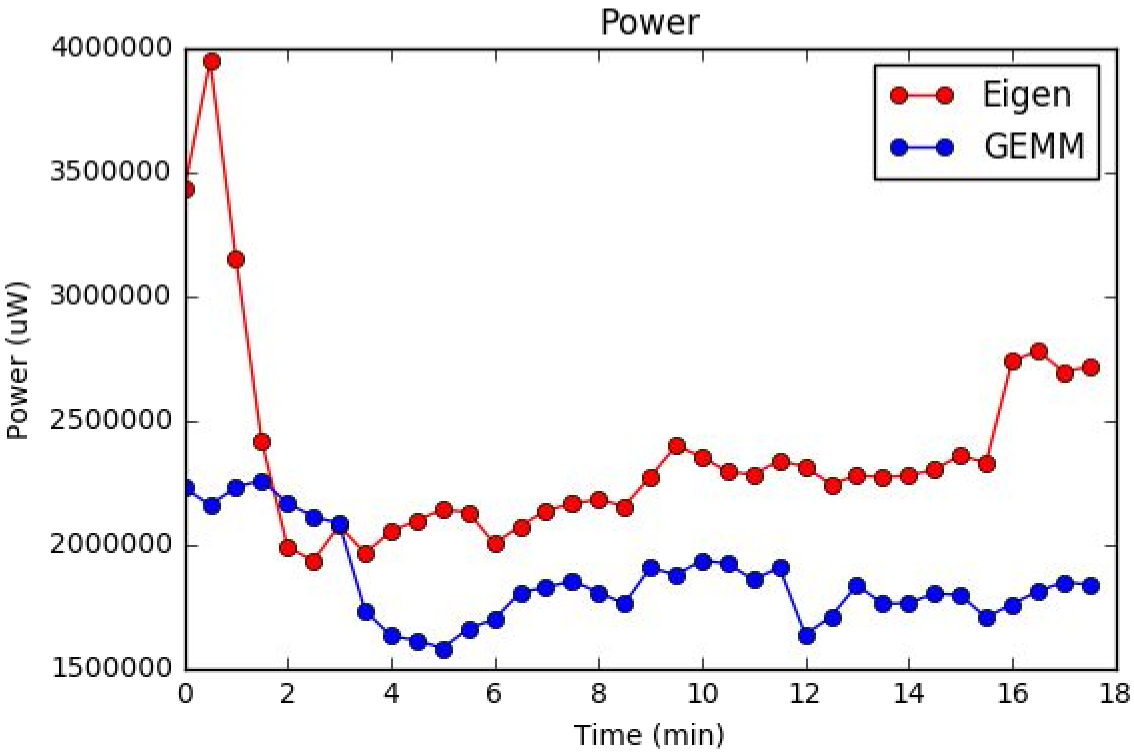
Eigen Inference Time: ~450ms

Op type	Count	Avg ms	Avg %
Conv2D	22	285	69.2
LRN	2	47.7	12.1
MaxPool	6	28.8	7.2
BiasAdd	24	19.2	4.8
Matmul	23	11.6	2.8
Relu	3	6.3	1.6
Concat	53	5.8	1.4

GEMMLowp Inference Time: ~900ms

Op type	Count	Avg ms	Avg %
Conv2D	57	350	42.9
Reqtz	116	97.1	11.1
Concat	9	85.4	10.2
MaxPool	13	78.7	9.4
LRN	2	54.3	6.4
BiasAdd	58	49.2	5.7
Range	116	40.1	4.5

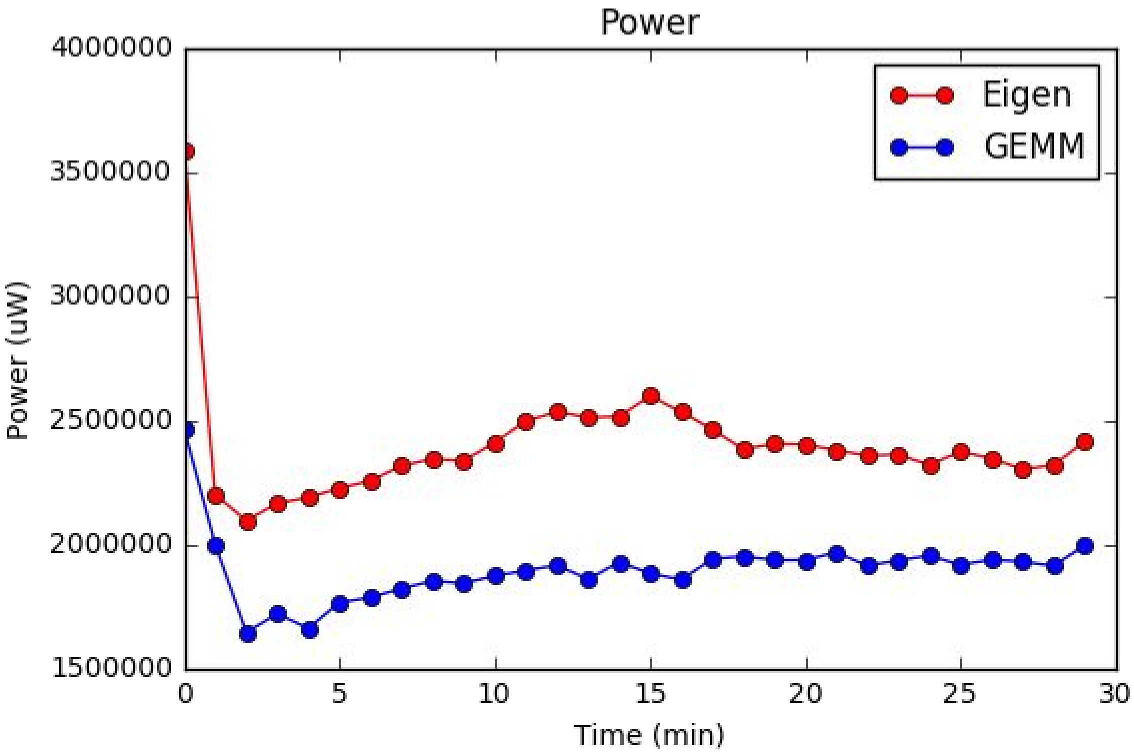
Evaluation - Power (3 * 6 min)



Eigen Average Power:
2367172 uW ~= 2.367W

GEMMLowp Average Power:
1856098 uW ~= 1.856W

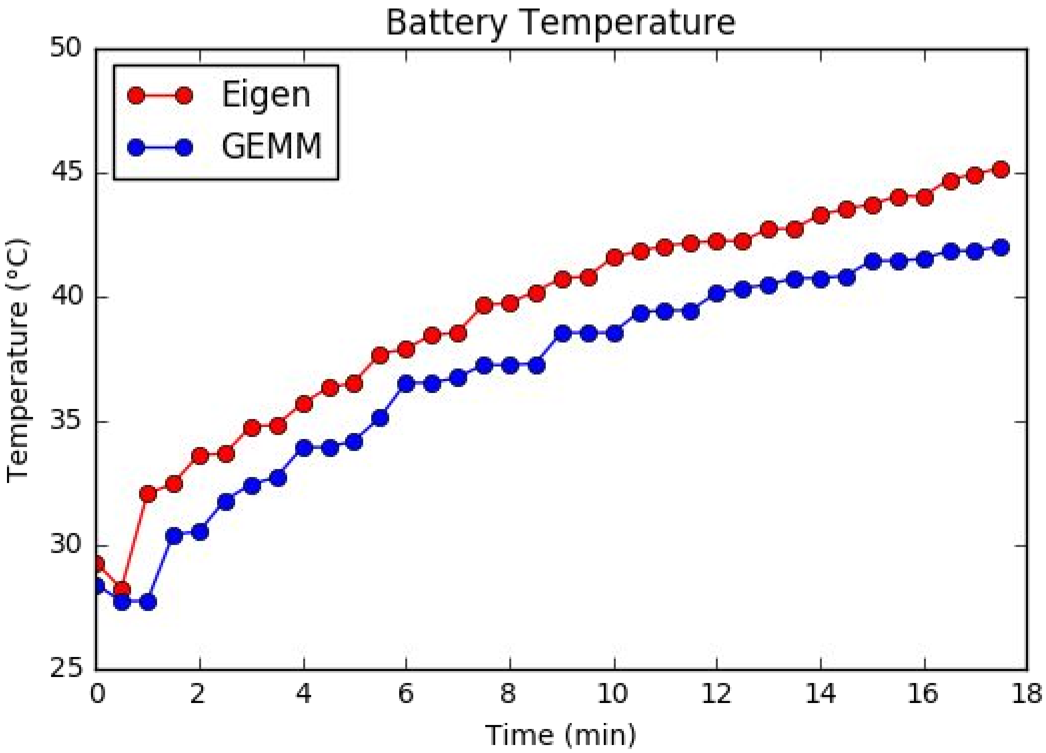
Evaluation - Power (30 min)



Eigen Average Power:
2403876 uW ~= 2.404W

GEMMLowp Average Power:
1898825 uW ~= 1.899W

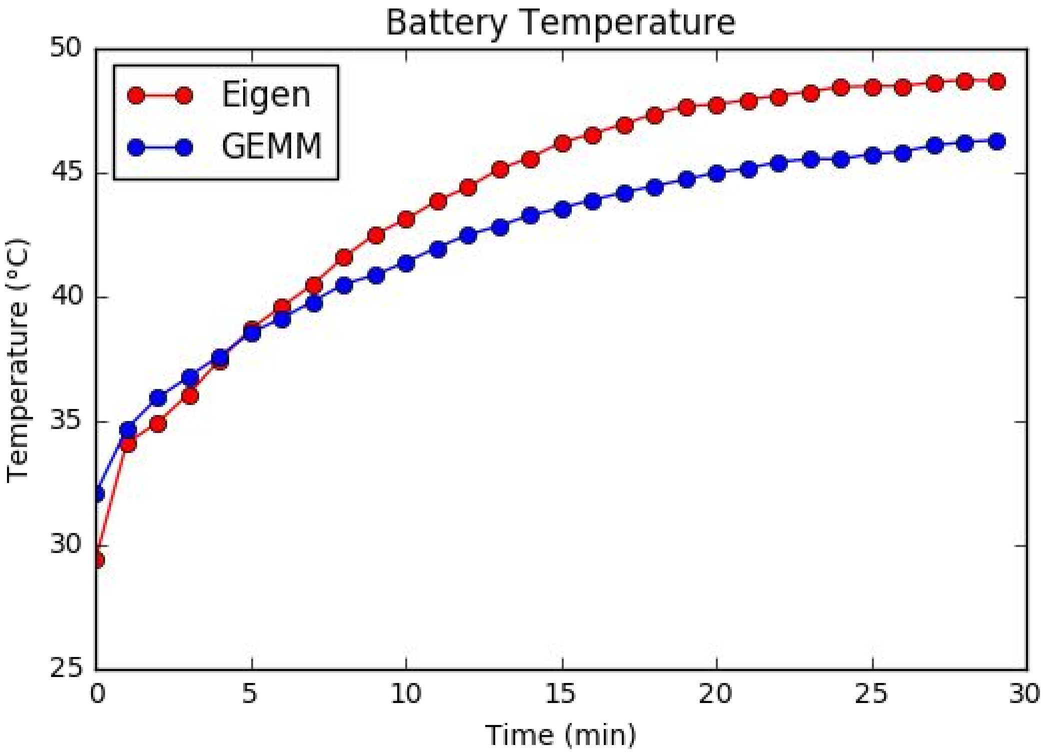
Evaluation - Battery Temperature (3 * 6 min)



Eigen Average Battery Temperature: 39.2 °C

GEMMLowp Average Battery Temperature: 36.8° C

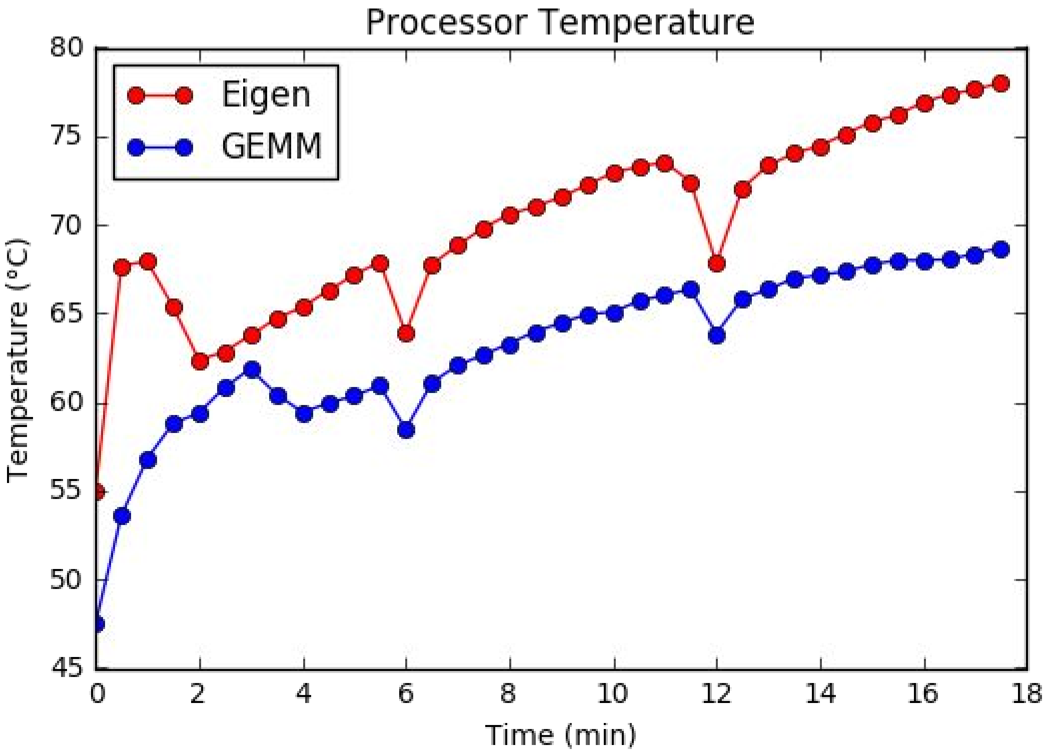
Evaluation - Battery Temperature (30 min)



Eigen Average Battery Temperature: 43.8 °C

GEMMLowp Average Battery Temperature: 42.1° C

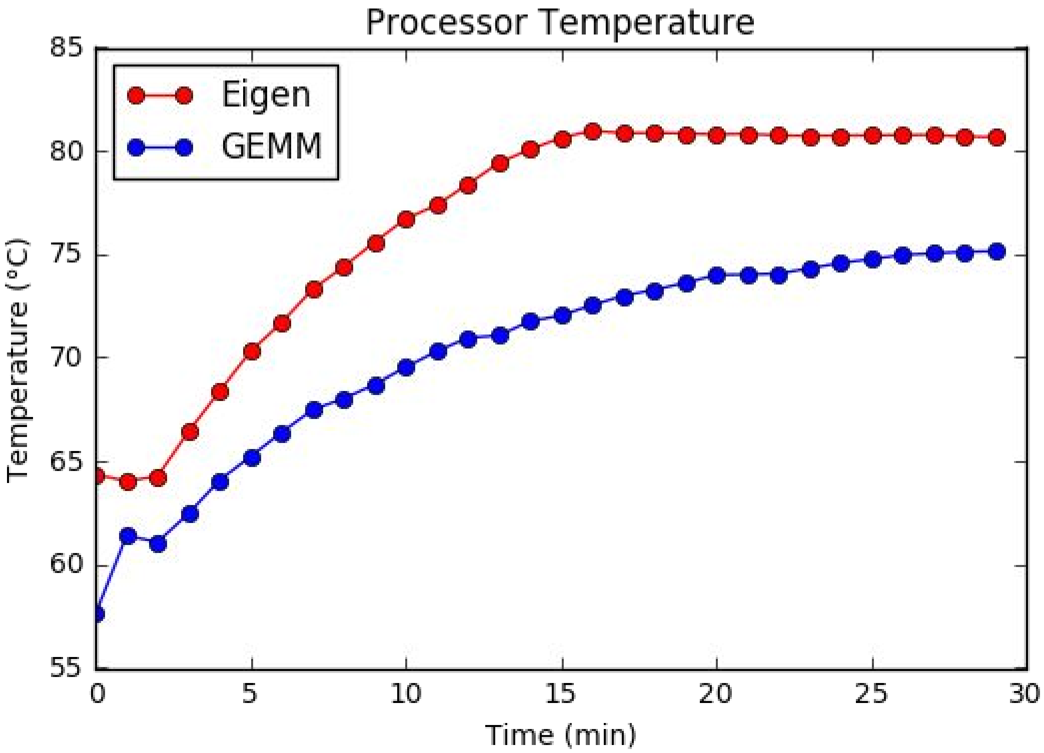
Evaluation - Processor Temperature (3 * 6 min)



Eigen Average Processor Temperature: 70.1 °C

GEMMLowp Average Processor Temperature: 63.1° C

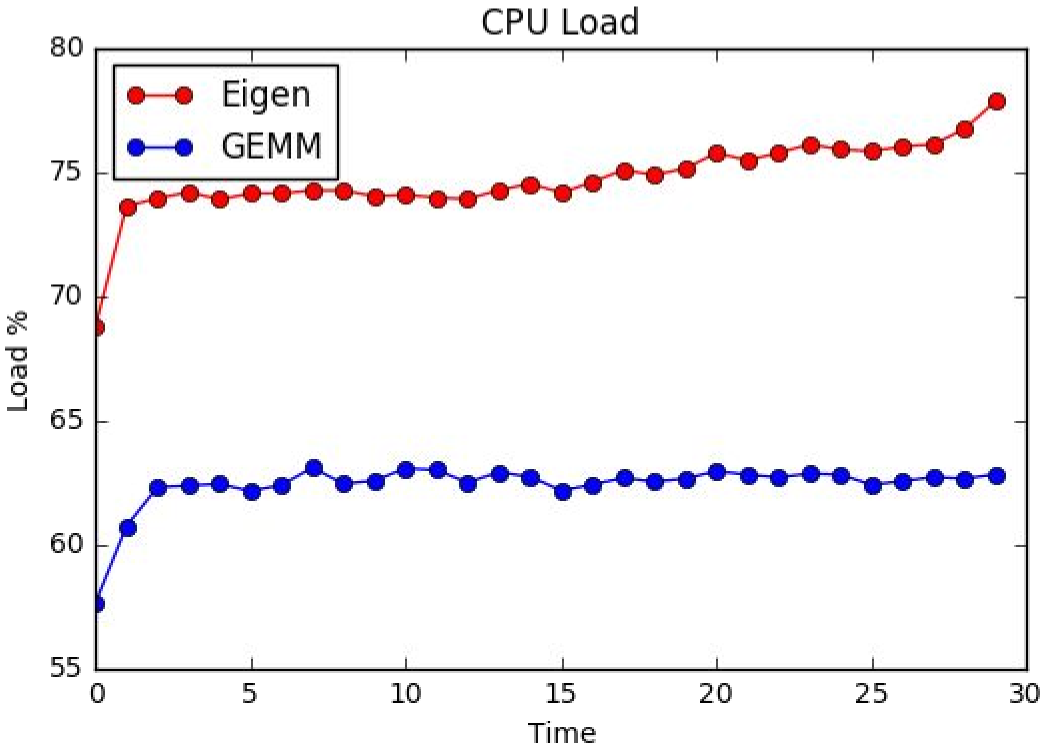
Evaluation - Processor Temperature (30 min)



Eigen Average Processor Temperature: 76.5 °C

GEMMLowp Average Processor Temperature: 70.2° C

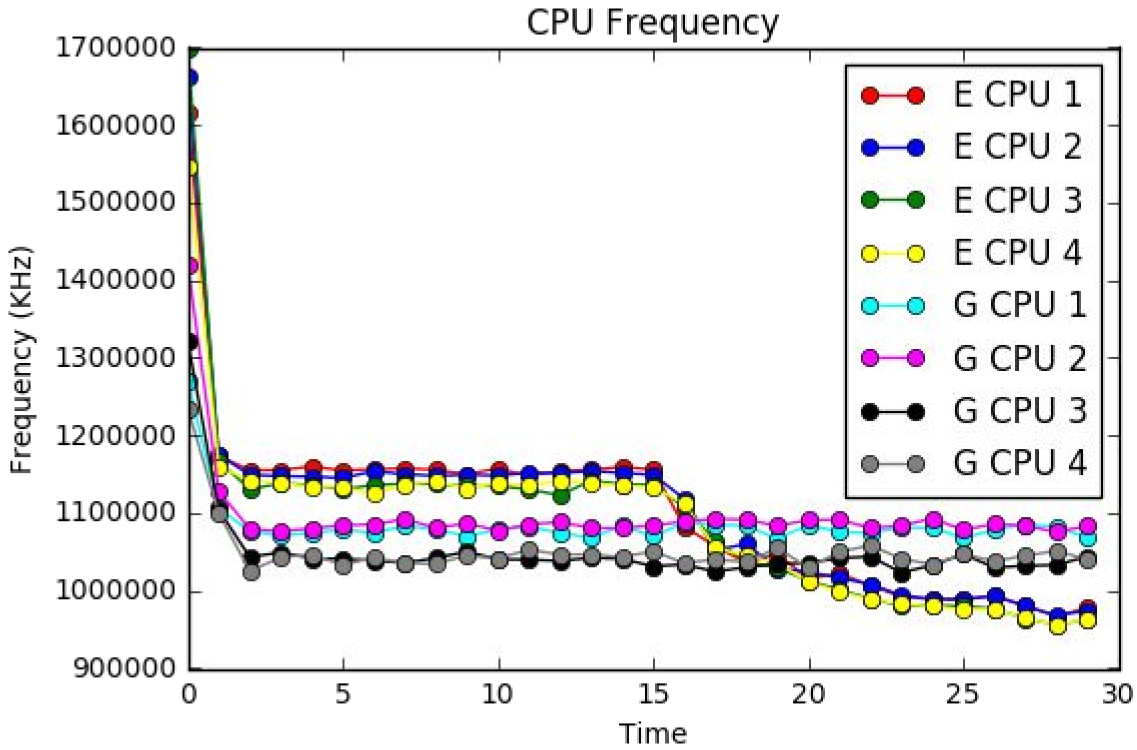
Evaluation - CPU Load (30 min)



Eigen Average CPU Load %: 74.7 %

GEMMLowp Average CPU Load%: 62.4 %

Evaluation - CPU Frequency (30 min)



Eigen Average CPU Freq:

Core 1: 1104187 KHz
Core 2: 1104544 KHz
Core 3: 1093375 KHz
Core 4: 1088829 KHz

GEMMLowp Average CPU Freq:

Core 1: 1084966 KHz
Core 2: 1097250 KHz
Core 3: 1049776 KHz
Core 4: 1050990 KHz

Conclusion & Future Work

- GEMMLowp, in general, contributes better performance than Eigen in Power Consumption, induced Battery Temperature and Processor Temperature and lastly, CPU Load %.
- RenderScript theoretically should be able to accelerate the Operations to achieve better performance; We have proved that RenderScript can successfully leverage the GPU on Android device.

- Future Work:
  - Use TF class to manage the static memory
  - Use RenderScript to accelerate Eigen/GEMMLowp
  - Bazel and RenderScript have too many bugs