3.2 TensorFlow

TensorFlow is a widely used framework for machine intelligence. It was originally developed and used by Google internally, until it was released as open-source project in 2015. TensorFlow represents a model computation as a dataflow model in the form of a directed graph. The graph is composed of a set of *nodes* that represent *operations* while *edges* between the nodes are *tensors* holding arbitrary dimensionality arrays of values. TensorFlow relies mainly on the `Eigen` [14] and `cuBLAS` [21] as a library for underlying linear algebra subroutines. On commodity Android devices, `Eigen` [14] is the library being used. While `Eigen` is very well optimized library for running on ARM processors using the ARM advanced SIMD instruction-set (`NEON`), it does not make use of other heterogeneous computing resources such as GPU and DSP.

Recently a cooperation between Google and Qualcomm has lead to adding Qualcomm Hexagon 682 DSP support to TensorFlow. Hexagon 682 DSP is an integrated part of the Snapdragon 835 SoC. According to official statement form Qualcomm, running TensorFlow on DSP is 25X times faster and 8X energy efficient than running on CPU. However, phones with Snapdragon 835 are not launched market yet.

3.3 RenderScript

Google introduced RenderScript [1] as a framework for running computationally intensive tasks at high performance on Android. RenderScript parallelizes the computation workloads across CPU cores and GPUs. It is commonly used to accelerate image processing and computer vision algorithms on mobile phones.

Developer express their data parallel tasks in terms of compute `kernels` with RenderScript code using a c-99 language in `.rs` files. RenderScript framework executes kernels in parallel across different data points and will distribute the execution across the available heterogeneous CPU cores and GPUs. During the build time, this code is compiled into an intermediate bytecode using `llvm` compiler. Android build tools also generate a reflected class with the name `ScriptC_`*`renderscript_filename`* for each `.rs` file. This class provides an interface to call the RenderScript functions from java/c++ code.

During the runtime on device, this bytecode is compiled again (just-in-time) into machine code using another compiler. The machine code is optimized for the device and is cashed so the just-in-time compilation happens only during the first time the code runs on device.

4. SYSTEM DESIGN

Running inferences using neural network model requires executing the forward pass of the model which involves different operations. We ran an experiment to decide which operations are more computationally expensive than others and hence it is more important to optimize their performance. In our experiment, we use the `TensorFlow` for Android library [4] to run the forward pass of inception [22] model on Nexus 6 phone. We observe the timing of every operation and of the whole model and compute the percentage of time spent running each operation type. The result shown in Figure 1 demonstrates convolution operations constitute the largest fraction of forward pass time (approx. 75%) while

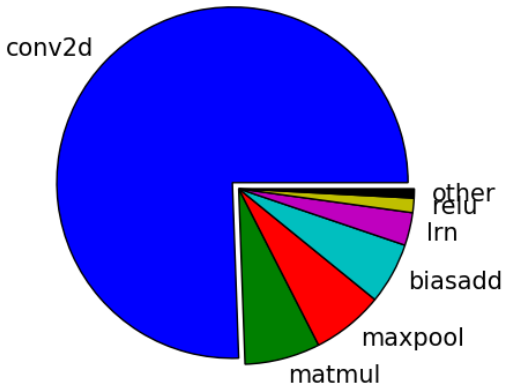

Figure 1: The time share of each type of operations during the forward pass of Inception model

matrix multiplication take the second largest fraction of the the forward pass time (approx. 7%). Therefore, we focus our efforts on these two kinds of operations. The following subsections discuss our approach to modify `TensorFlow` to run these operations using `RenderScript` instead of the default `Eigen` ARM `NEON`-based implementation.

4.1 Matrix Multiplication (`MatMul`)

Matrix multiplication operation (`MatMul`) is an essential ingredient in all kinds of deep learning models as fully connected layers require matrix multiplication between the input matrix and the weight matrix. Fortunately, matrix multiplication is easy to parallelize as every element in the output matrix can be computed independently from other elements. Therefore, it can benefit a lot from data-parallel execution. `RenderScript` framework has built-in implementation for multiple matrix BLAS (Basic Linear Algebra Subprograms) operations defined in `ScriptIntrinsicBLAS` class. We modified `TensorFlow` to make use of the `RenderScript` implementation of matrix multiplication instead of the default `MatMul` implementation that uses `Eigen` library.

4.2 Convolution Operation (`Conv2D`)

Convolution operations are the core building block of CNN models such as the inception model [22] which has 22 convolution layers. Convolution layer consists of a number of filters (their values are learned during the training phase). The input and output of each convolution layer are represented as volumes where the depth of volume represents the number of feature maps. For the first input layer, depth will be 3 the RGB color channels. Convolution layer applies the sliding filters across the height and width of input volume transforms it into another volume with the new set of feature maps.

In order to parallelize the convolution operation using `RenderScript`, we developed a render script kernel file to execute the computation of output values in the convolution output volume in parallel.

5. EXPERIMENTS

We implemented `RSTensorFlow` by extending the `TensorFlow` v1.0.1 implementation. We used Nexus 6 and Nexus 5X mobile phones. Both phones are running Android 7.0 (Nougat, API 24). The two phones have different CPU and GPU models. The detailed hardware specifications of the two phones are shown in Table 1.

Model	Nexus 5X	Nexus 6
SoC	Snapdragon 808	Snapdragon 805
Processor	1.8GHz (8 cores)	2.7 GHz (4 cores)
GPU	Adreno 418	Adreno 420
Memory	2 GB	3 GB

Table 1: Hardware specs of phones used in our experiments

5.1 Matrix Multiplication Operation Results

We benchmark the running time of the modified matrix multiplication operation and compare it against the running time of the default `Eigen`-based implementation on the two different phones. We perform matrix multiplication between square-matrices of different sizes and measure the time for each multiplication. The timing result are shown in Figure 2.

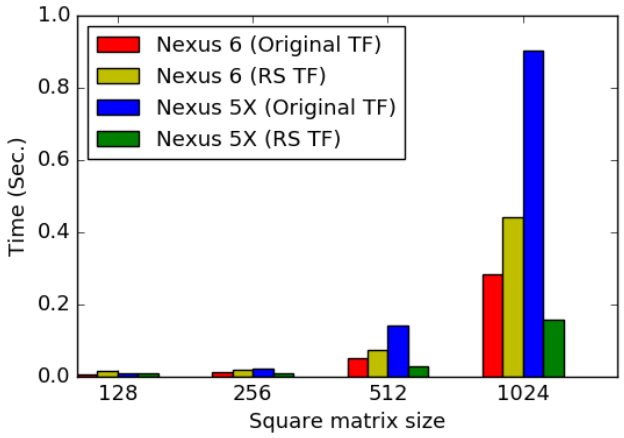

Figure 2: Time of matrix multiplication between square matrices using Original TensorFlow and RenderScript TensorFlow

The results of our matrix multiplication experiments show that on Nexus 5X the RenderScript implementation becomes significantly faster as the matrix size increases. When square matrix size = 1024, matrix multiplication using RenderScript took 158 milli-seconds which is *6 times* faster than the default implementation using `Eigen` library that took 904 milli-seconds. However, on Nexus 6 phone the RenderScript-based matrix multiplication was slower than the default `Eigen`-based implementation. The reason is the following. `RenderScript` does not provide the user with control (nor guarantees) about which hardware resource will be used to perform the computation. By monitoring the CPU and GPU frequencies during the experiment, we observed that on Nexus 5X the GPU was used to perform the matrix multiplication which explains the speed-up we obtained, while on Nexus 6 RenderScript did not use the GPU and used only two cores of the available 4 CPU cores while `Eigen` fully utilized the four CPU cores. This explains why `RenderScript` matrix-multiplication was faster than `Eigen` on Nexus 5X and slower than it on Nexus 6. We repeated our experiments multiple times of different phones of the same type and observed similar results.

5.2 Convolution Operation Results

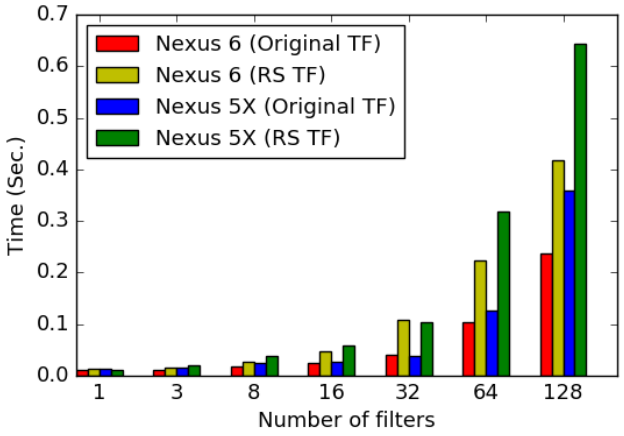

Figure 3: Time of applying convolution operation on input image with size 224x224 using Original TensorFlow and RenderScript TensorFlow

We also benchmarked the running time of both our RenderScript-based implementation of the convolution operation and compared it against the running of the default `Eigen`-based implementation. Our benchmarking results are shown in Figure 3. Unfortunately, we have not noticed speed up in the performance of convolution operation. Even on the Nexus 5X phone were RenderScript utilized the GPU, convolution operation was slower than the default TensorFlow implementation based ARM `NEON` acceleration. This might be due to the memory overhead associated with using RenderScript that required copying data from/to special buffers (referred to as allocations in RenderScript). Therefore, optimizing the runtime of convolution operation on mobile GPU than the `Eigen`-based implementation remains an interesting research question.

5.3 ConvNet Model Results

We also studied the effects of our RenderScript extension of TensorFlow on the total runtime of the forward pass of the inception [22] convolutional neural network model for image recognition. Inception model consists of 22 convolution and pooling layers and two large fully connected layers at the end of the model computation graph. It recognizes the input image as one of 1000 class labels of the ImageNet [11] dataset.

The results of our benchmarking experiments are shown in Table 5.3. We observe significant speed up when utilizing `RenderScript` to perform the matrix multiplication (approximately *3 times faster* on Nexus 5X). On the other hand, using our implementation of the convolution operations in

	Nexus 6			Nexus 5X		
Batch size	Original TF	TF + RS MatMul & RS Conv2D	TF + RS MatMul	Original TF	TF + RS MatMul & RS Conv2D	TF + RS MatMul
1	0.453	1.765	0.312	0.699	2.775	0.351
2	0.718	1.757	0.370	1.235	2.782	0.471
3	0.979	1.879	0.475	1.785	2.811	0.649
4	1.246	1.841	0.575	2.335	2.853	0.839
5	1.535	1.830	0.645	2.988	2.930	1.025

Table 2: Comparison of the time (in seconds) required to run the forward pass of Inception model using the original TensorFlow v1.0.1, TensorFlow with `RenderScript` matrix multiplication and convolution operations, and TensorFlow with `RenderScript` matrix multiplication only.

`RenderScript`, tends to bring the whole model execution time slower than the original TensorFlow implementation.

5.4 RNN Activity Recognition Results

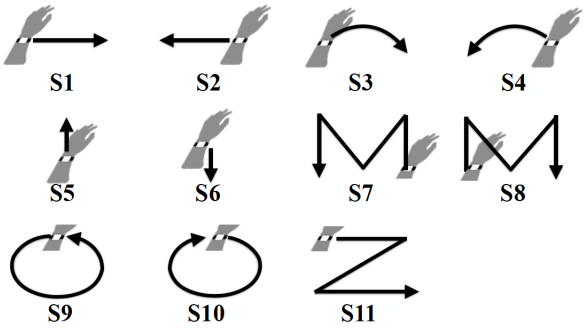

Figure 4: List of hand gestures used in our LSTM-based activity recognition model

We also benchmarked the performance of running recurrent neural network sequence classification model using Long-Short Term memory (LSTM) units. We developed an activity recognition RNN model consisting of 512 LSTM units that receive as input a time-series (100 time-steps) of accelerometer and gyroscope sensor measurements to identify the hand gesture as one of the 11 signs which are shown in Figure 5.4. The model can achieve above 90% classification accuracy evaluated using 5 folds cross validation on a dataset of 1290 examples collected by four persons.

Evaluation results shown in Figure 5.4 shows that `RSTensorFlow` slightly improves the classification runtime. Notice, that time-series classification using RNN is a computationally expensive task as the state update for every time step involves several matrix multiplication operations.

The result shows that although the `RSTensorFlow` is slower than the original `TensorFlow` model when the model runs on a single example, `RSTensorFlow` becomes faster than the original `TensorFlow` when we increase the batch size (as a result of increasing the size of the matrix multiplication).

6. CONCLUSION

In this paper, we introduced `RSTensorFlow` an accelerated deep learning framework on commodity android devices using the heterogeneous computing framework `RenderScript`. Although, we noticed that GPU was not used by `RenderScript` on all phone models. When GPU is used, `RSTensorFlow` improves matrix multiplication operations a lot and therefore we observed significant speedup in running different models on Nexus 5X phones. Optimizing other deep learning operations and profiling the energy costs of running on CPU vs GPU are potential future research directions.

Acknowledgement

This research was supported in part by the NIH Center of Excellence for Mobile Sensor Data-to-Knowledge (MD2K) under award 1-U54EB020404-01, and by the U.S. Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-16-3-0001. Any findings in this material are those of the author(s) and do not reflect the views of any of the above funding agencies. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

7. REFERENCES

[1] Android renderscript API guide. https://developer.android.com/guide/topics/renderscript/compute.html.

[2] Caffe android library. https://github.com/sh1r0/caffe-android-lib.

[3] gemmlowp: a small self- contained low-precision GEMM library.

[4] Tensorflow android demo. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android.

[5] Torch android. https://github.com/soumith/torch-android.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[7] AMD. Core math library (ACML). *URL http://developer. amd. com/acml. jsp*, page 25, 2012.

[8] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS*

	Nexus 6		Nexus 5X	
	`TensorFlow`	`RSTensorFlow`	`TensorFlow`	`RSTensorFlow`
batch size=1	0.824	3.986	1.314	2.213
batch size=64	9.456	9.234	7.698	7.253

Table 3: Comparison of the time (in seconds) required to run the LSTM sequence classification model.

2011, BigLearning Workshop, Granada, Spain, volume 3. Citeseer, 2011.

[9] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[10] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning, 2016.

[13] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[14] G. Guennebaud and B. Jacob. Eigen 2 matrix library. *nd [Online]. Available: http://eigen. tuxfamily. org.*

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] N. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, and F. Kawsar. Dxtk: Enabling resource-efficient deep learning on mobile and embedded devices with the deepx toolkit. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, pages 98–107. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[18] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pages 1–12. IEEE, 2016.

[19] S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi. Cnndroid: GPU-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1201–1205. ACM, 2016.

[20] M. Motamedi, D. Fong, and S. Ghiasi. Fast and energy-efficient cnn inference on iot devices. *arXiv preprint arXiv:1611.07151*, 2016.

[21] C. Nvidia. Cublas library. *NVIDIA Corporation, Santa Clara, California*, 15:27, 2008.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.