Project Update

-Anisha Aswani

New Slides from Slide no. 90

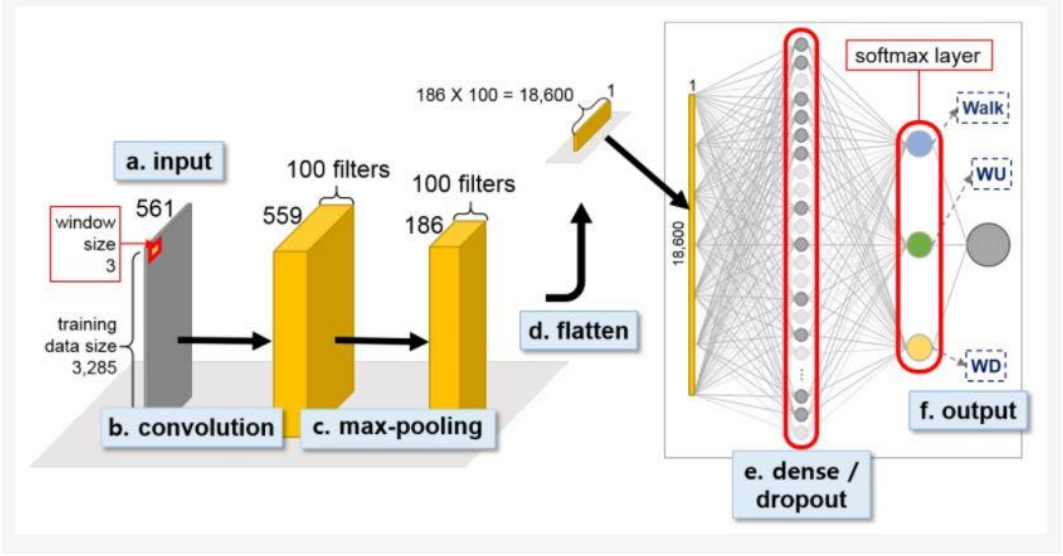
- Read about Tensorflow , Tensorflowlite, and Caffe.
- Studied the tutorial on image classification present on the Tensorflow website. Ran the complete project to understand the working of Tensorflowlite and its interface with Android application.
- Studied into quantization and binarization, the basics and the application of these techniques in a machine learning model.
- Studied pre-trained model for image classification and its effect on a regular training model.
- Worked on course content of ECE 561 which gave me a good base for quantization and binarization.
- Completed a literature survey on Optimization Techniques to reduce memory footprints.
- From the literature survey, I got to know about other techniques of optimization like pruning and use of sparsity.

Introduction to HAR Model

- Looked into the Human Activity Recognition dataset and recent papers based on HAR.
- Researched into multiple papers and selected two papers with one having 1D CNN model and other having LSTM-RNN model.
- Selected a CNN model presented by Heeryon Cho and Sang Min Yoon in the paper “ Divide and Conquer-based 1D CNN Human Activity Recognition Using Test Data Sharpening ” and its code implementation presented in the github repository https://github.com/heeryoncho/sensors2018cnnhar.

Understanding the paper based on 1D CNN

- The paper is based on Divide and Conquer based 1D CNN where there are 2 stages of recognition: abstract activity recognition (dynamic, static) and individual activity recognition(walk, walk-up, walk-down or sit, stand, lay).
- The first stage uses a decision tree classifier to classify dynamic or static activity.
- The second stage uses a 1D CNN model having the following layers.



Understanding the paper based on 1D CNN

- This paper implements signal sharpening process on the test data.
- A Gaussian filter is applied which has the effect of attenuating high frequency signals. The σ and αparameters determines the degree of attenuation.
- The test dataset is divided into 2 sets to obtain a range for σ and α. The accuracy of the model is plotted for different σ and α values.

Model Parameters and Results

- Static Model parameters

```
+++ DATA STATISTICS +++

train_static shape:  (4067, 561)
test_static shape:  (1560, 561)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 559, 30)           120
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 557, 50)           4550
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 555, 100)          15100
_________________________________________________________________
flatten (Flatten)            (None, 55500)             0
_________________________________________________________________
dense (Dense)                (None, 3)                 166503
_________________________________________________________________
dropout (Dropout)            (None, 3)                 0
=================================================================
Total params: 186,273
Trainable params: 186,273
Non-trainable params: 0
```

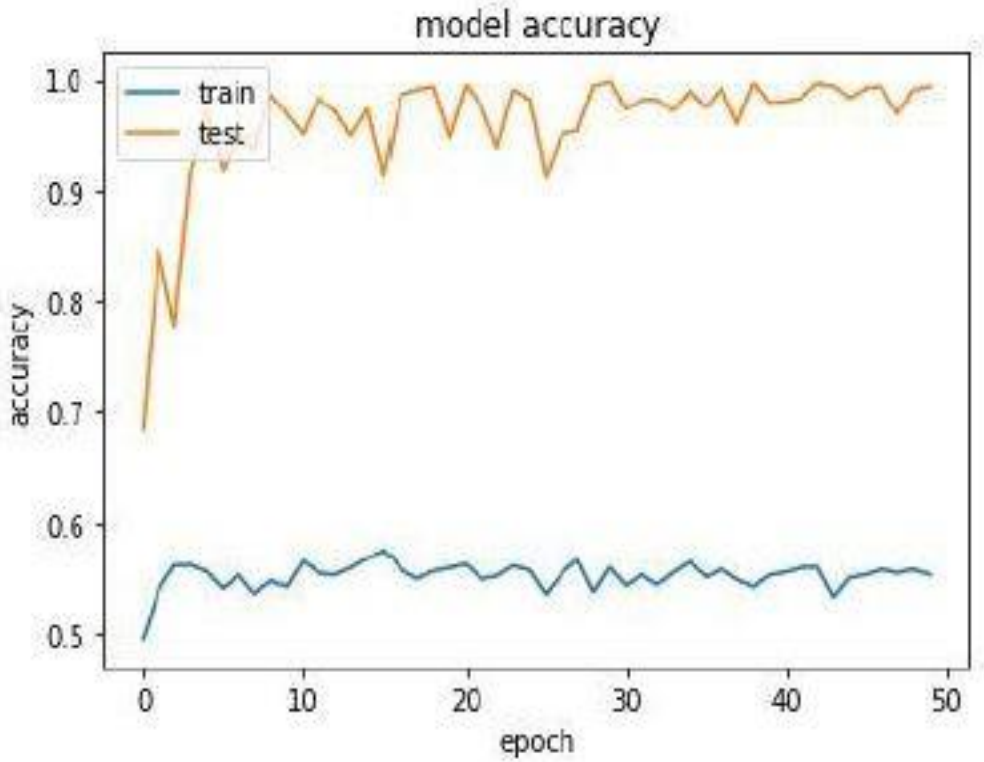
Dynamic Model Parameters

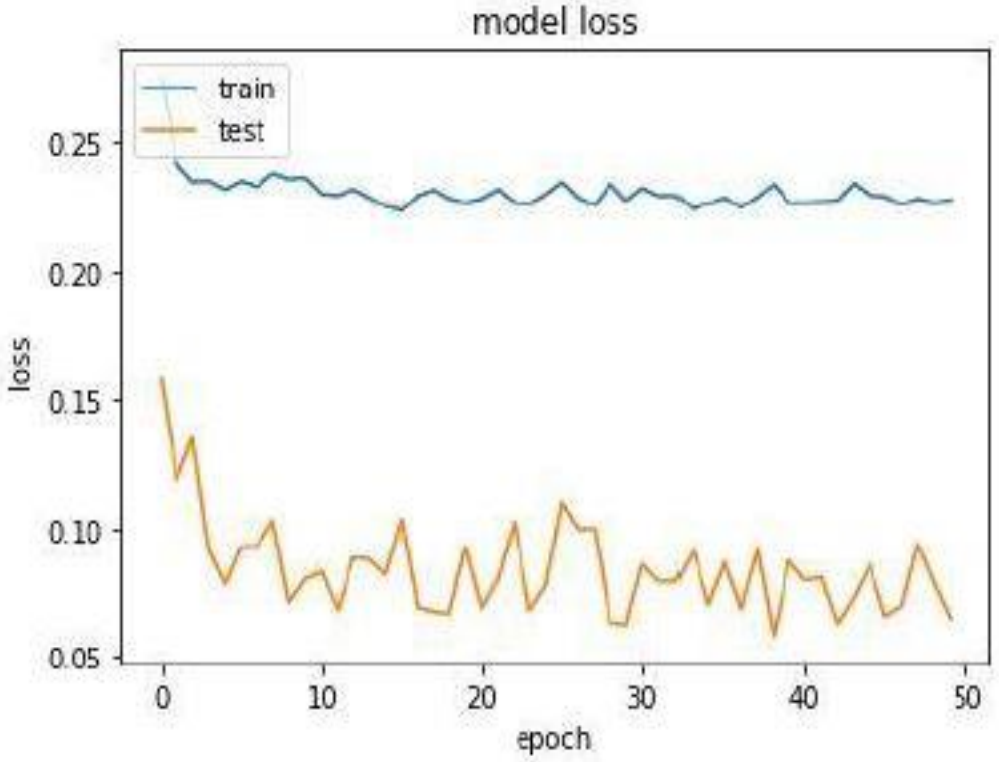
```
+++ DATA STATISTICS +++

train_dynamic shape:  (3285, 561)
test_dynamic shape:  (1387, 561)
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tenso
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 559, 100)          400
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 186, 100)          0
_________________________________________________________________
flatten (Flatten)            (None, 18600)             0
_________________________________________________________________
dense (Dense)                (None, 3)                 55803
_________________________________________________________________
dropout (Dropout)            (None, 3)                 0
=================================================================
Total params: 56,203
Trainable params: 56,203
Non-trainable params: 0
```

Results

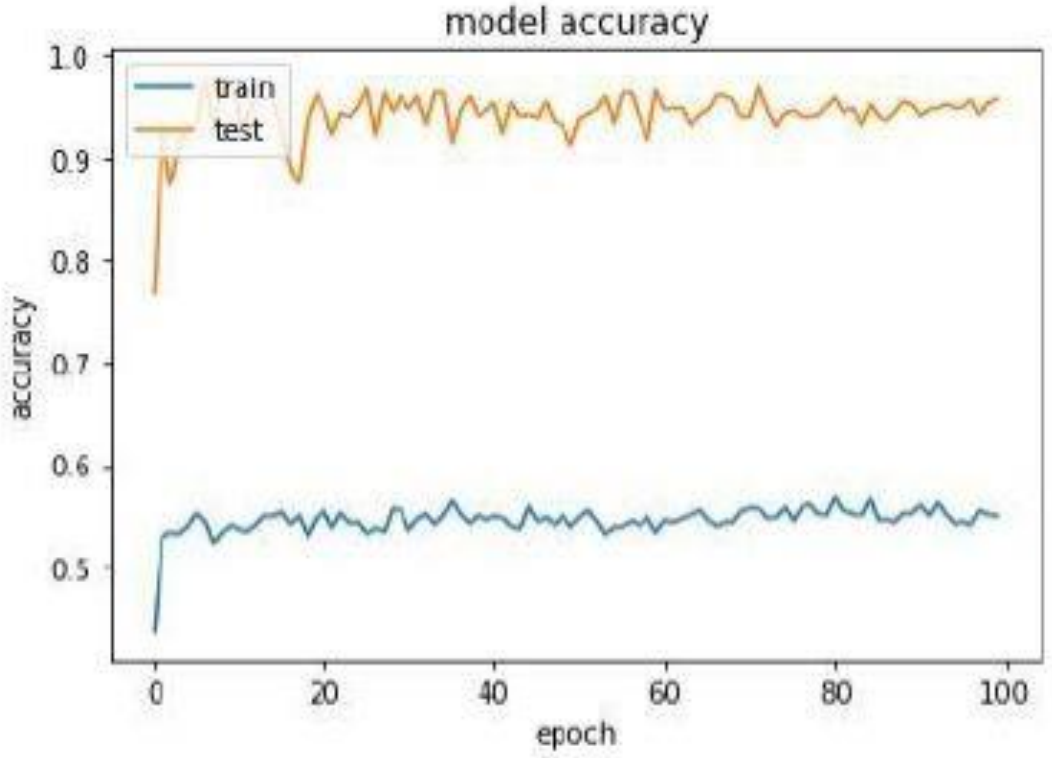
- Dynamic accuracy : Train accuracy 55%,Validationaccuracy98%

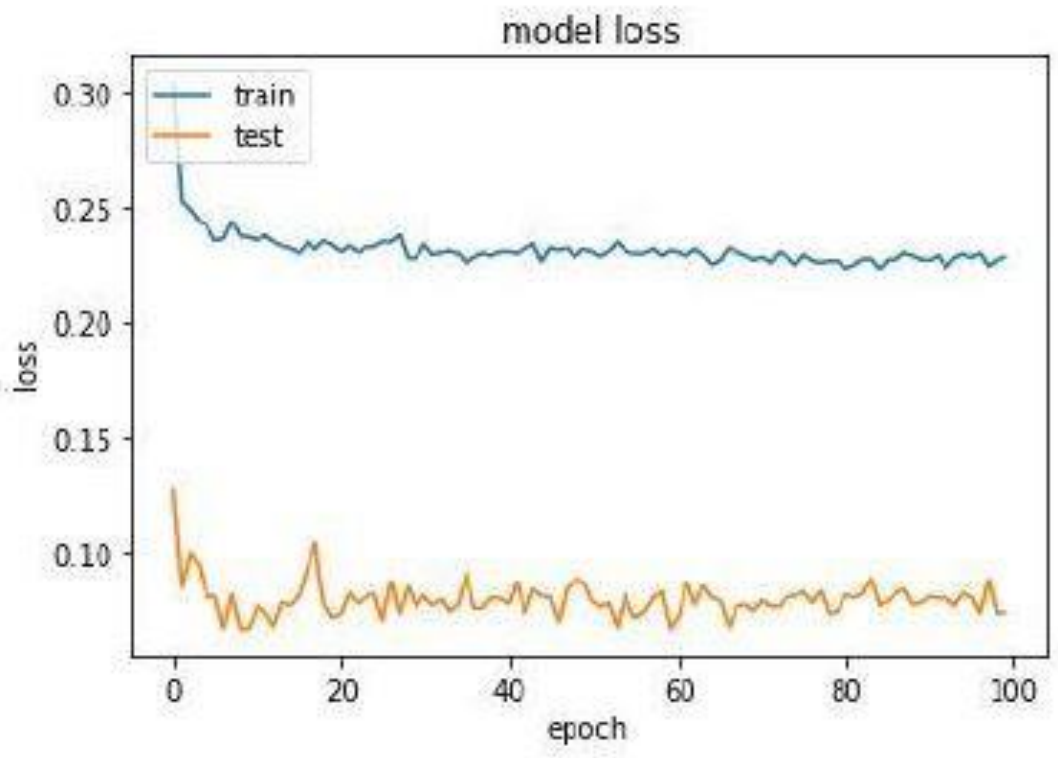




Results Contd.

Accuracy: Train accuracy 54%, Validation accuracy 99%





Comparison of regular and sharpened data

```
-----------------------------------
      NO SHARPEN ACCURACY
-----------------------------------
0.9473684210526315
[[206  39   0]
 [  2 264   0]
 [  0   0 268]]
```

```
-----------------------------------
      SHARPENED  ACCURACY
-----------------------------------
0.9731113956466069
[[226  20   0]
 [  1 265   0]
 [  0   0 269]]
```

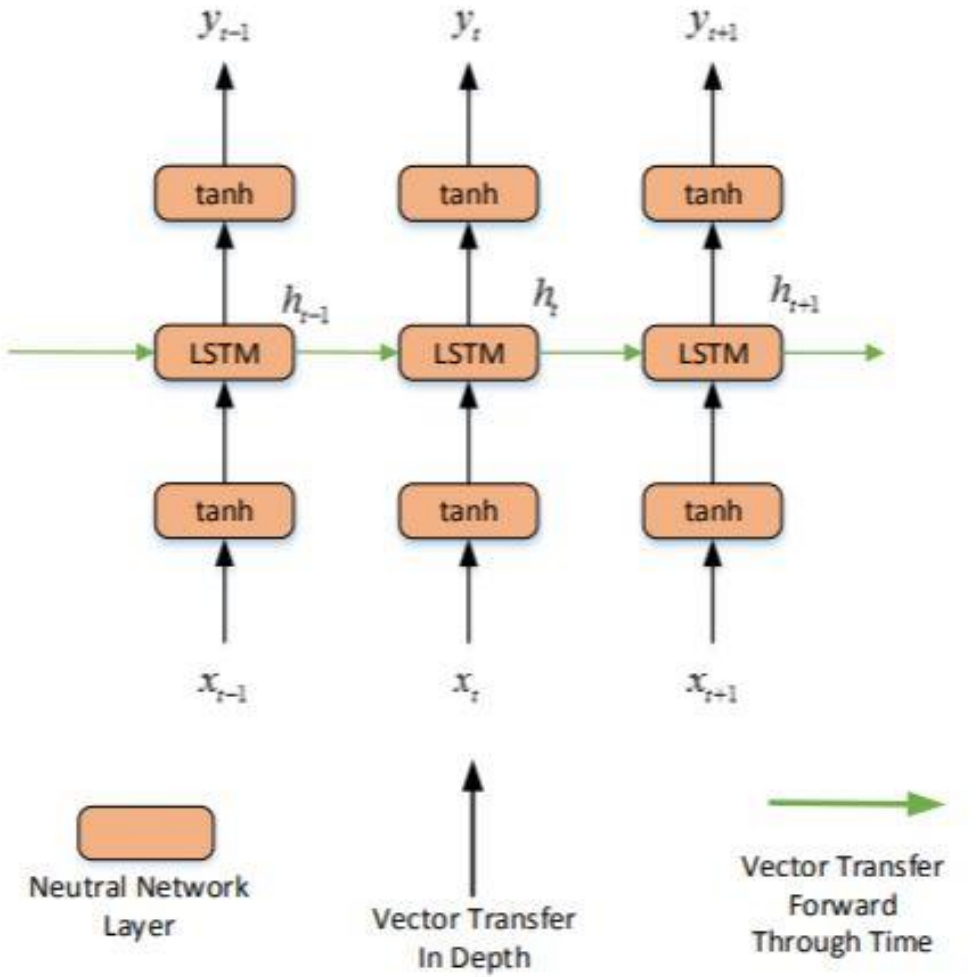
Understanding Paper Based on LSTM-RNN

- To get more insights on different models based on HAR, I selected a paper that implements LSTM-RNN. (https://arxiv.org/abs/1708.08989) (Github: https://github.com/guillaume-chevalier/HAR-stacked-residual-bidir-LSTMs)
- With this model, I learned more about the working of LSTM and RNN model.
- The code implements LSTM network, Bidirectional LSTM network, and a LSTM-RNN model.

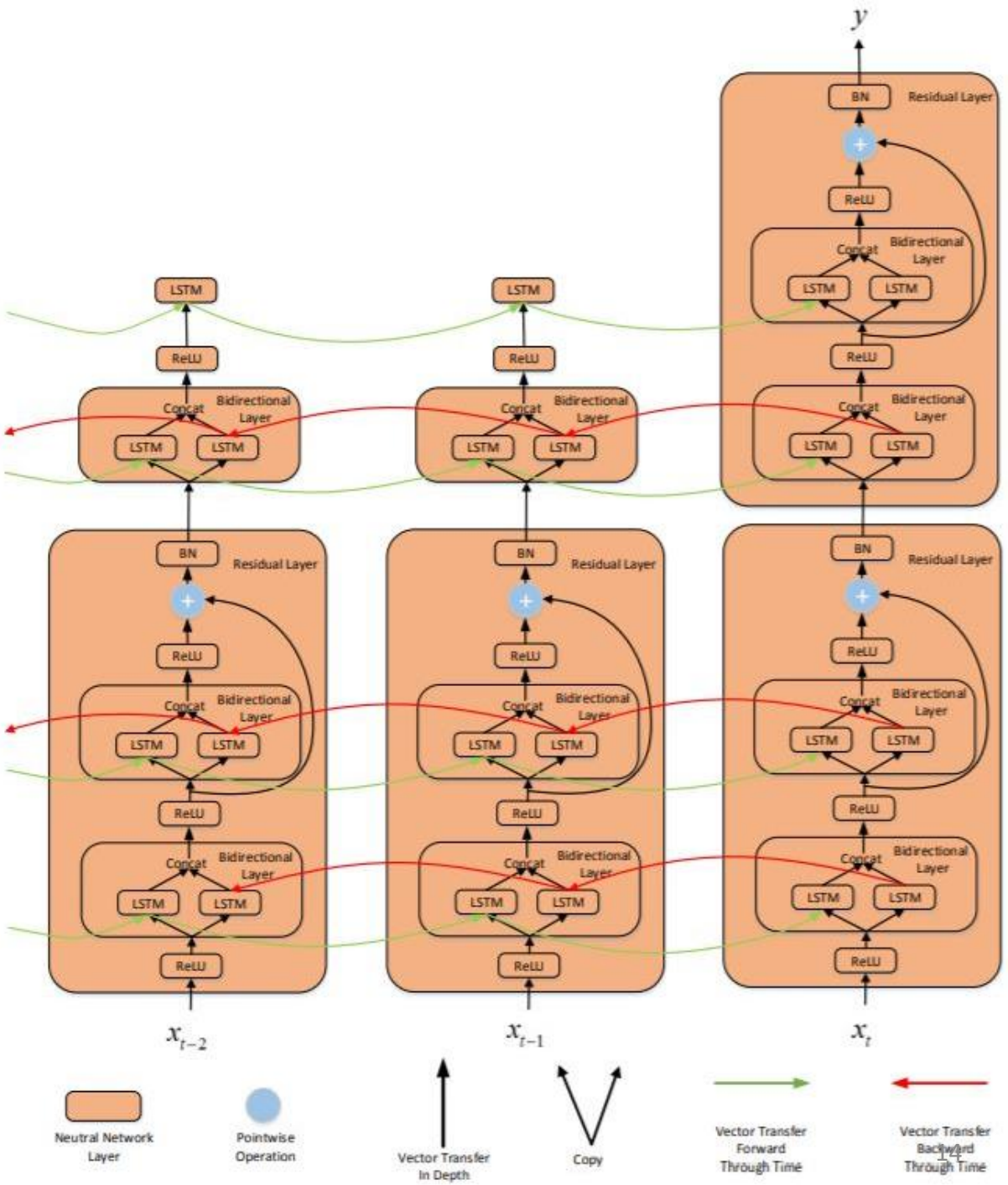
Understanding Paper Based on LSTM-RNN

- Executed the code available for the paper, “Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors”. https://github.com/guillaume-chevalier/HAR-stacked-residual-bidir-LSTMs)
- The paper presents a deep network architecture using residual bidirectional long short-term memory cells.
- A comparison of results is done between baseline LSTM model and residual bidirectional LSTM model.

Basic LSTM model



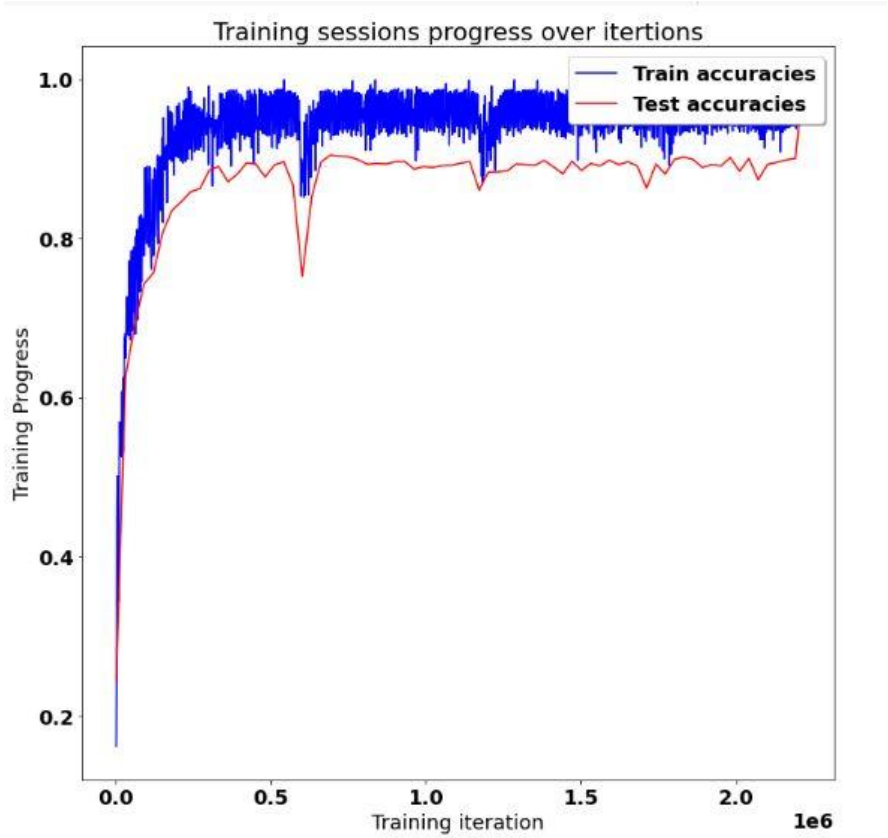
Residual Bidirectional LSTM model



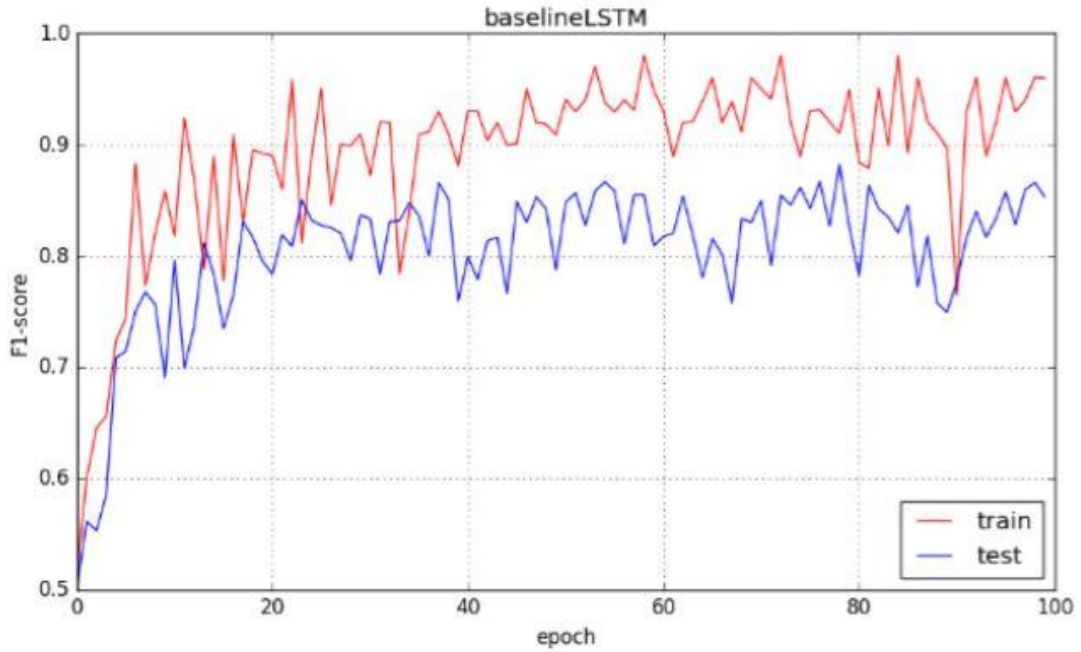
Architecture of proposed model

- Understood the working of basic LSTM model and residual bidirectional LSTM Model.
- There are 2 hidden layers having residual connection.
- Each residual layer contains 2 bidirectional layers.
- The network has 2x2 architecture which means it has 8 LSTM cells in total.
- Activity function used is ReLU.
- The state of neural network is reset at each new window for each new prediction.

Results of Baseline LSTM

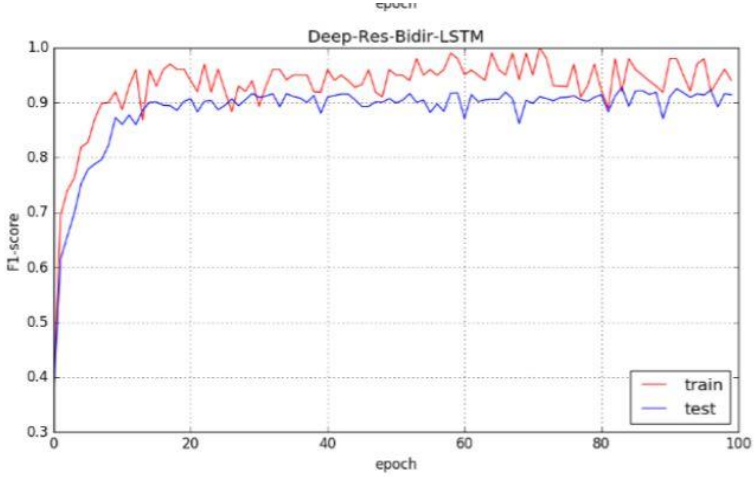


Results obtained from code

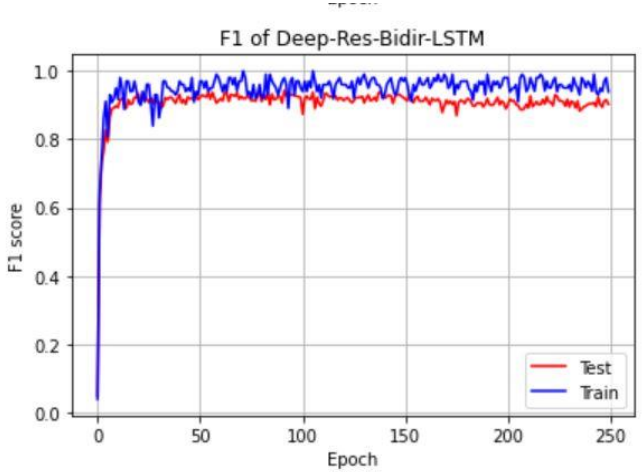


Results in published paper

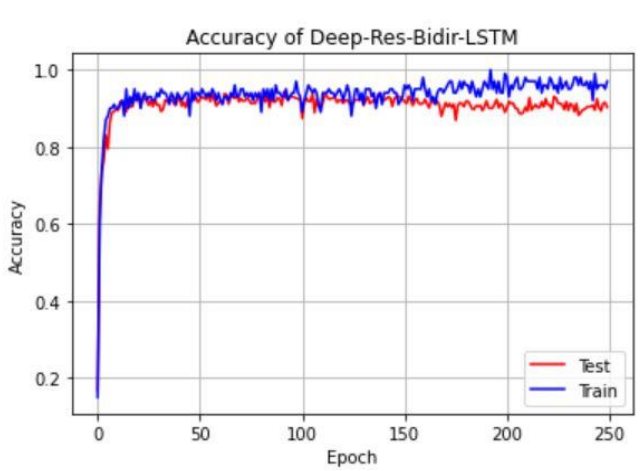
Results for Residual Bidirectional LSTM



Accuracy plot from paper



F1 accuracy obtained from code



Accuracy obtained from code

Result of Baseline LSTM

```
Testing Accuracy: 89.7862255732727%

Precision: 89.9374404906154%
Recall: 89.78622327790974%
f1_score: 89.77971717711846%
```

F1 score is a weighted average of accuracy and recall which ranges between 0 and 1. It is defined as:

$$F_1 = 2\sum_c \frac{N_c}{N_{total}} \frac{prec \times recall}{prec + recall}$$

Where N_c is sample count of class *c*, and N_{total} is the total sample count of the dataset.

Result of Residual Bidirectional LSTM

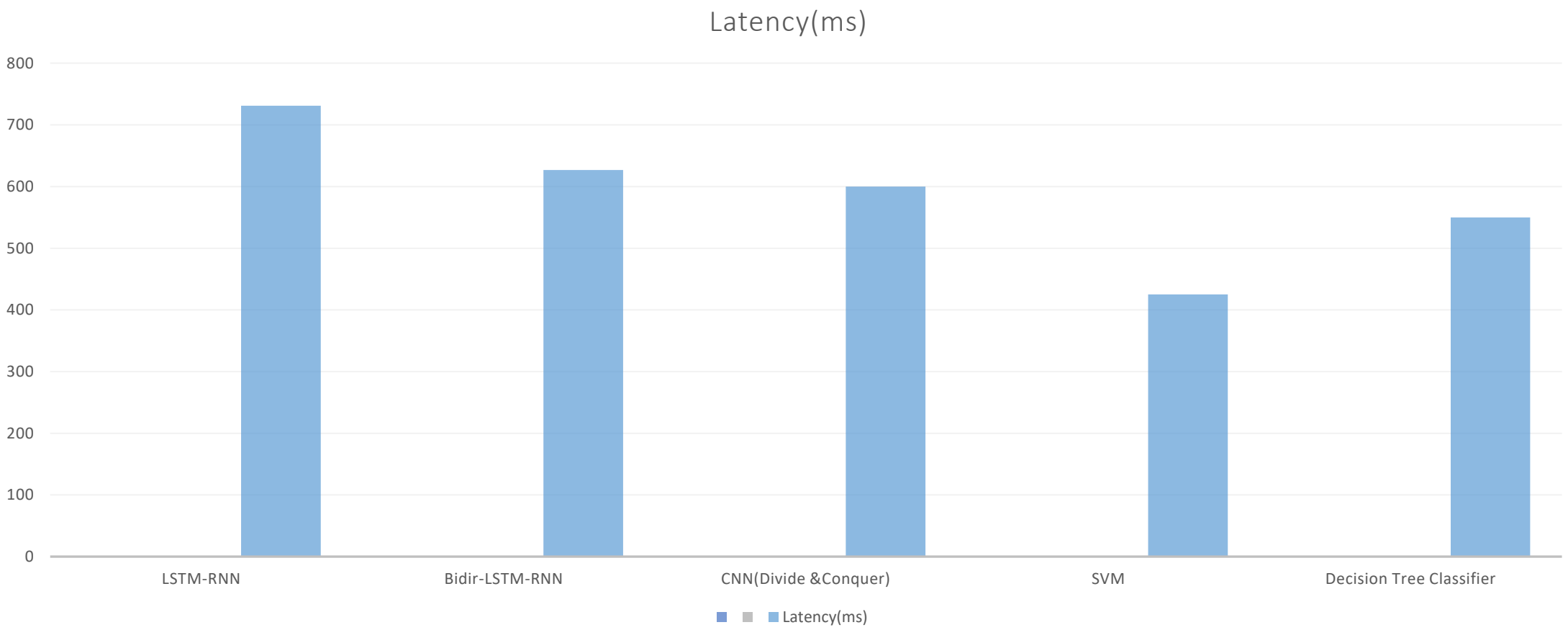
```
final test accuracy: 0.9029521346092224
best epoch's test accuracy: (0.94299287, 'iter: 91')
final F1 score: 0.9029107241557373
best epoch's F1 score: (0.9429314519857301, 'iter: 91')
```

Conclusion on LSTM model paper

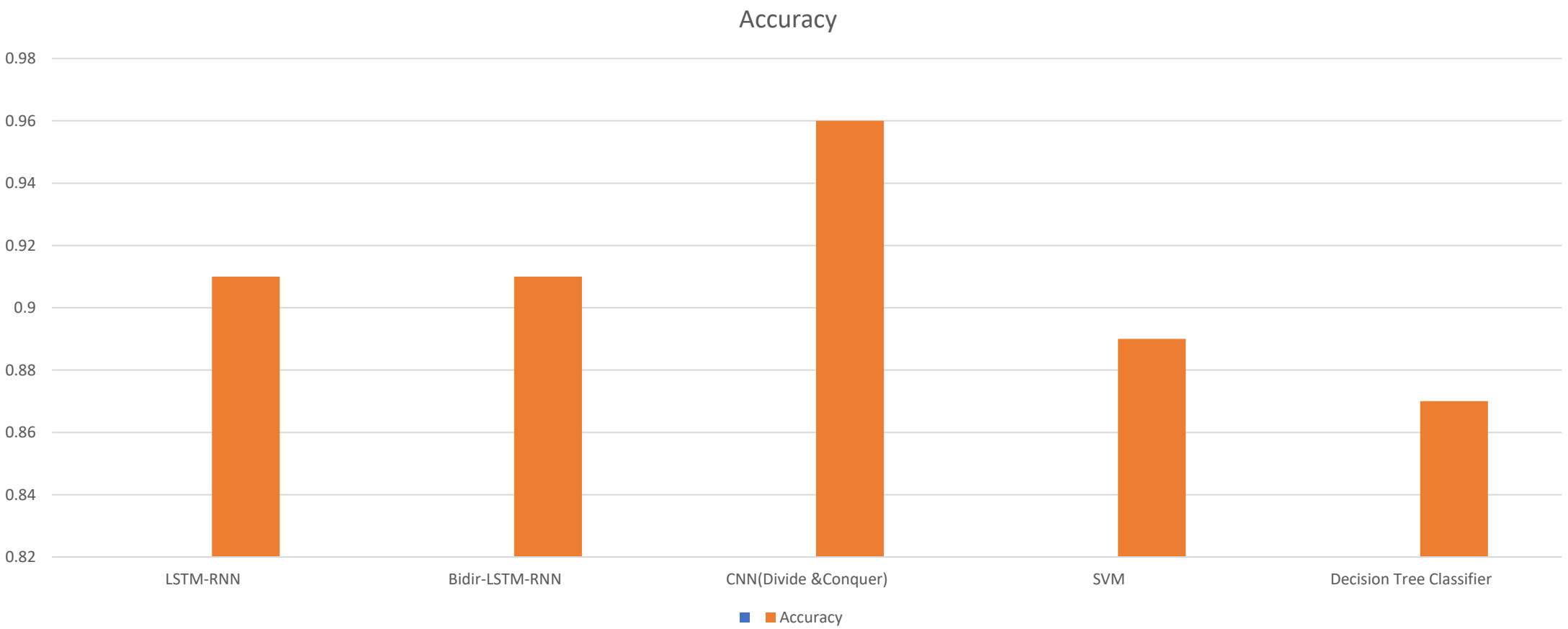
- It is observed that Residual Bidirectional LSTM architecture on HAR dataset gives better accuracy results of 94% than the baseline LSTM mode (accuracy 89%).
- The drawback is the large usage of memory and high latency for training the model due to the complexity of its deep architecture.

Reference:
Yo Zhao, Rennong Yang, Guillaume Chevalier, Maoguo Gong, “Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors”, arXiv:1708.08989v2

Comparison of Accuracy and Latency of all models



Accuracy Plots



Basics of CNN Model

- To get a better understanding of sequential model I started building my own basic CNN model.
- Building this model gave me a better understanding of the effect of different layers, hidden layers and kernel filters on number of parameters and accuracy of the model.
- Completed an online coursera course (https://www. understanding of the coursera.org/specializations/deep-learning) to brush up on the math behind CNN models and Sequential models.

Android Application

- Over the summer, I started looking into the android application on which the compressed ML model was to be deployed.
- Used the Github repository https://aqibsaeed.github.io/on-device-activity-recognition to understand the working of the complete project.
- Tried to execute the android application presented in the repository but came across a million configuration issues that were mostly because of different versions of android studio.
- Recently completed he tutorial of android app development on Kotlin and working on building my application for Human Activity Recognition.

Optimization of ML Model

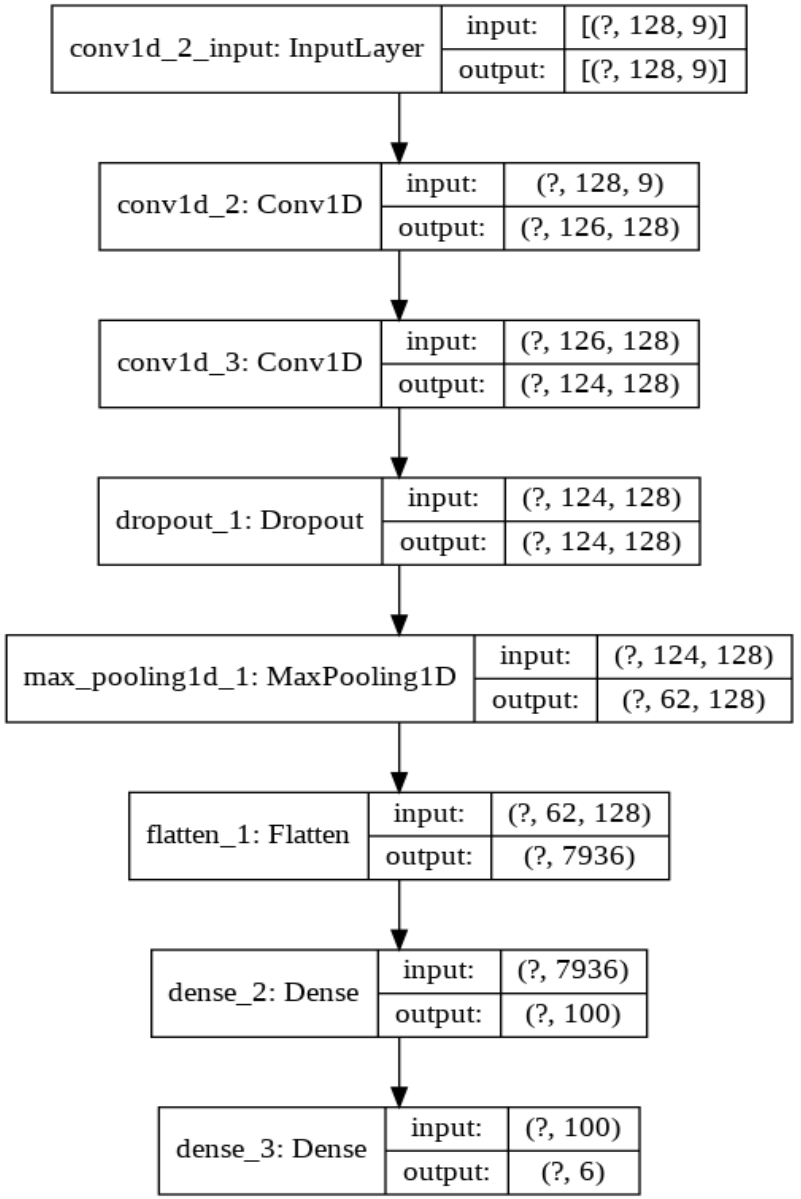
- Researched on PCA and LDA and their affect on CNN model for model accuracy ,latency, and memory consumed.
- Looked into the effect of principle component analysis and linear discriminant analysis on CNN model.
- Used a basic CNN architecture to understand the use of PCA and LDA.
- Tried to implement PCA and LDA on the baseline CNN model and achieved.

Optimization: Pruning

- Tried a Tensorflow example for pruning with Keras. https://github.com/tensorflow/model-optimization/blob/master/tensorflow_model_optimization/g3doc/guide/pruning/pruning_with_keras.ipynb
- Applied weight pruning to the HAR CNN model.
- Model had initial sparsity of 50% and end sparsity of 80%.
- Compressed the model using Tensorflow Tflite converter. Also tried post training quantization.
- Results are as follows.

Baseline CNN model with model summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 126, 128)          3584
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 124, 128)          49280
_________________________________________________________________
dropout_1 (Dropout)          (None, 124, 128)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 62, 128)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7936)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               793700
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 606
=================================================================
Total params: 847,170
Trainable params: 847,170
Non-trainable params: 0
_________________________________________________________________
```



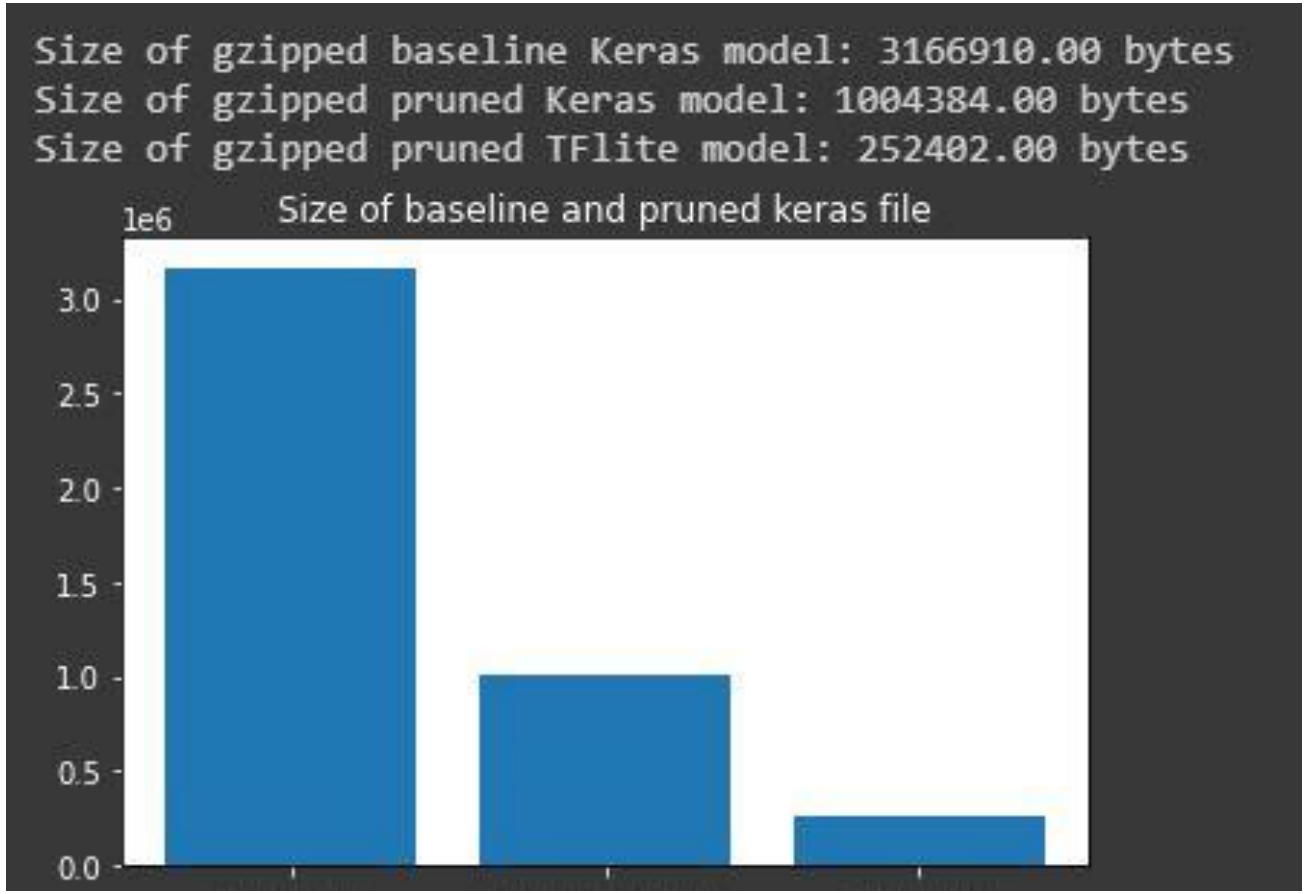
Pruned model summary

Baseline model accuracy: 91%
Pruned model accuracy: 94%

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
prune_low_magnitude_conv1d_2 (None, 126, 128)          7042
_________________________________________________________________
prune_low_magnitude_conv1d_3 (None, 124, 128)          98434
_________________________________________________________________
prune_low_magnitude_dropout_ (None, 124, 128)          1
_________________________________________________________________
prune_low_magnitude_max_pool (None, 62, 128)           1
_________________________________________________________________
prune_low_magnitude_flatten_ (None, 7936)              1
_________________________________________________________________
prune_low_magnitude_dense_2  (None, 100)               1587302
_________________________________________________________________
prune_low_magnitude_dense_3  (None, 6)                 1208
=================================================================
Total params: 1,693,989
Trainable params: 847,170
Non-trainable params: 846,819
_________________________________________________________________
```

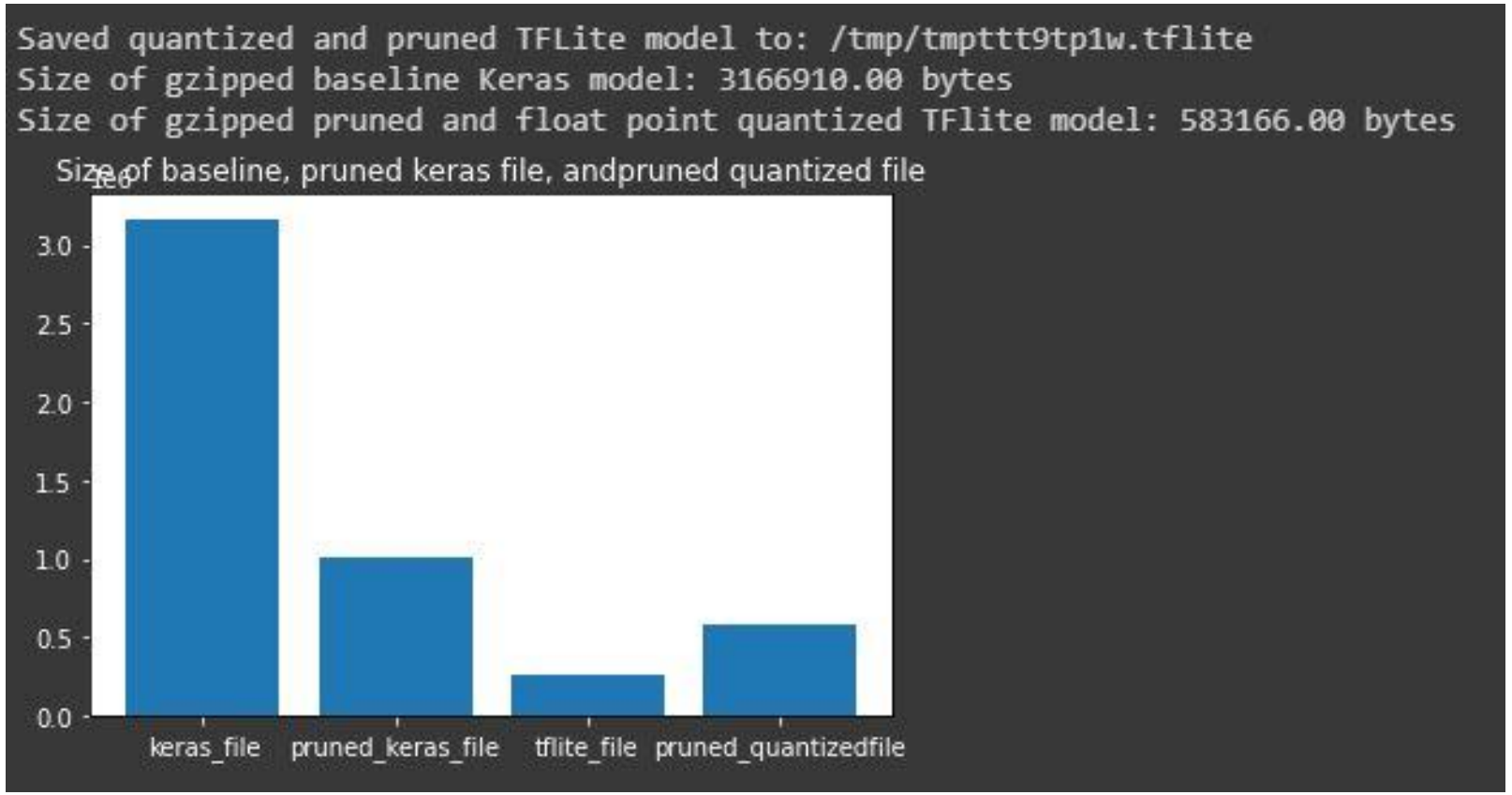
Converting keras model to Tflite model

- Used Tensorflow TFLiteconverter to convert the sequential model.
- Achieved the following model sizes for baseline model, pruned model, and tflite model.



Post training quantization

- Tried post training quantization (float 16) on the pruned model.
- Achieved the following model size.



Accuracy

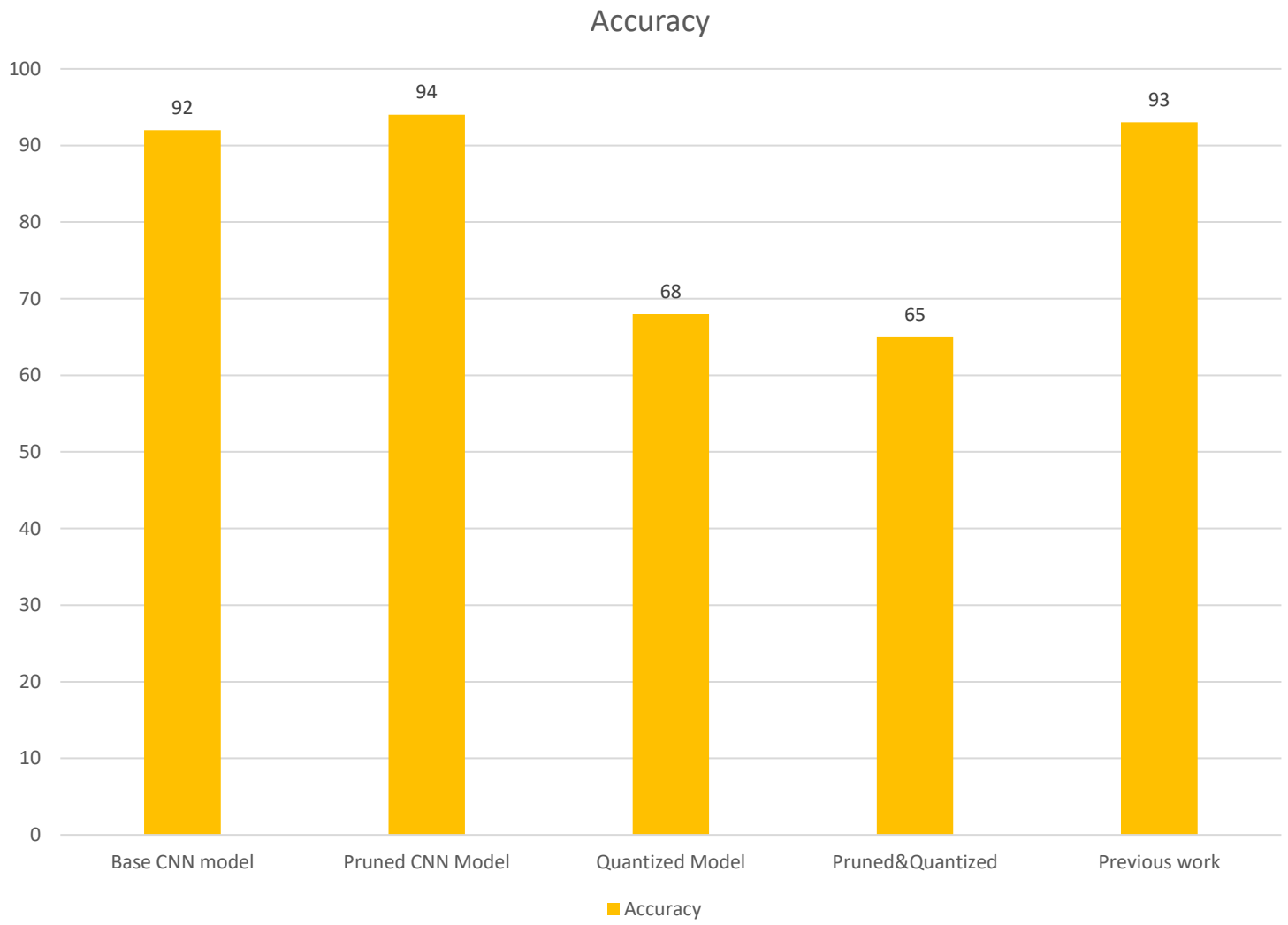
- The pruning and quantization are good methods for compressing models.
- The drawback of this big size compression is of model accuracy.
- Tflite model accuracy dropped by 20% and the test accuracy achieved was 70% as compared to pruned model accuracy (94%) and baseline model accuracy (91%).

Model Properties

- Accuracy plot of base CNN model, pruned CNN model, Quantized CNN model with no pruning, Quantized CNN model with pruning.
- All the above are compared to the CNN model of previous work which uses the Divide and Conquer method for HAR Classification. The previous work is based on first classifying the activity as static or dynamic and then further classifying the respective activities for static and dynamic activities. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5949027/

Accuracy

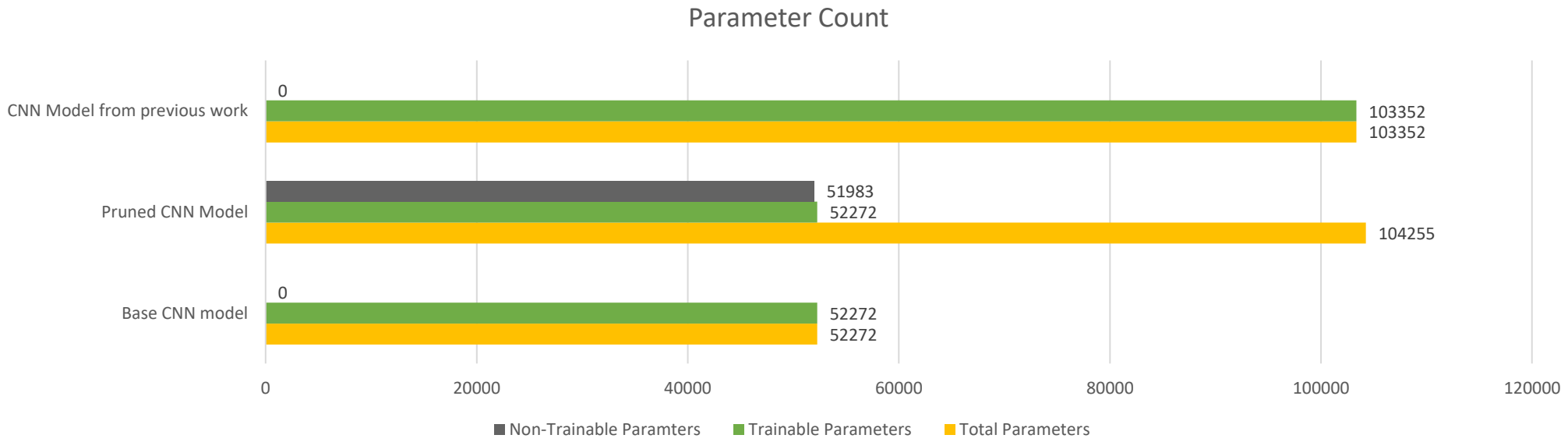
Model	Accuracy
Base CNN model	92
Pruned CNN Model	94
Quantized Model	68
Pruned & Quantized	62
Previous work	93



Parameter Count

- The parameter count for the base CNN model, Pruned CNN model, and previous work of Divide and Conquer model is compared.

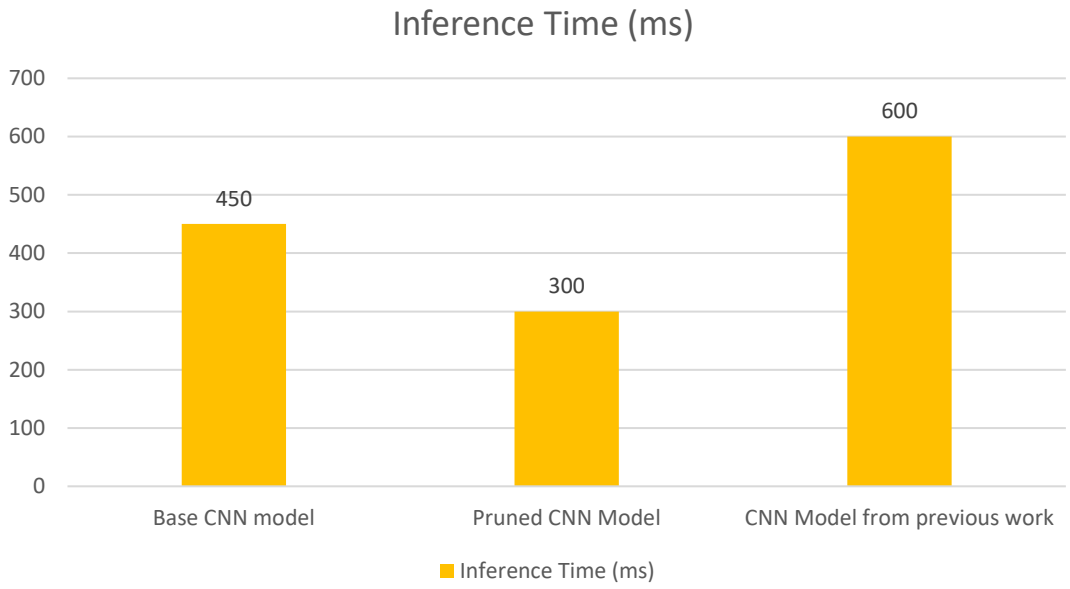
Model	Total Parameters	Trainable Parameters	Non-Trainable Paramters
Base CNN model	52272	52272	0
Pruned CNN Model	104255	52272	51983
CNN Model from previous work	103352	103352	0



Inference Time

- The inference time is calculated for the base CNN mode, pruned CNN model, and previous work of Divide and Conquer CNN model.
- Tried to get inference time for tflite model by running the android app but the app is still giving configuration problems.

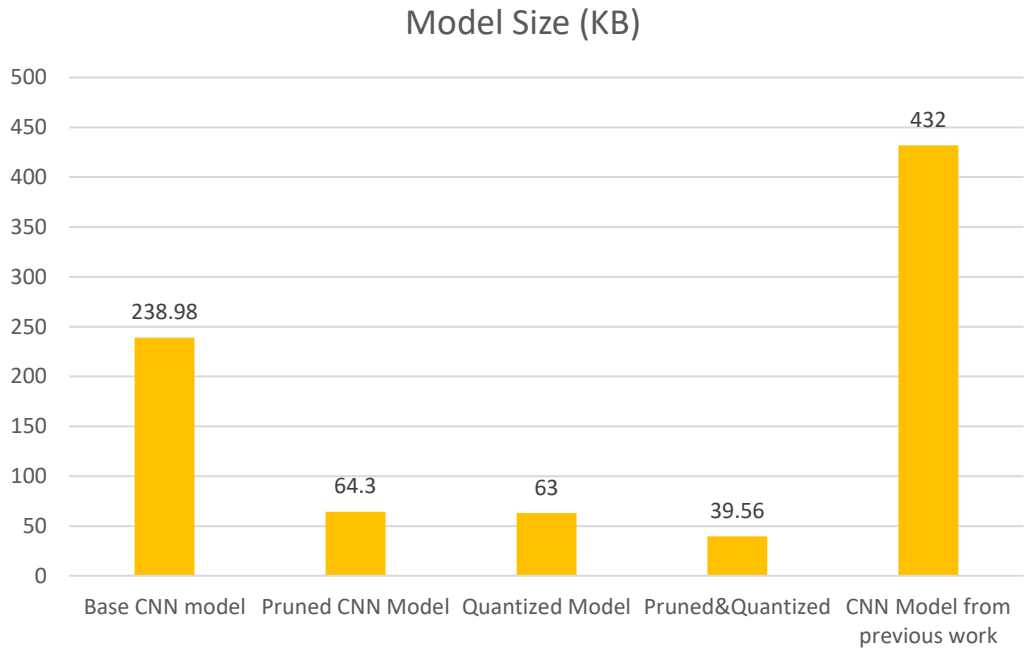
Model	Inference Time (ms)
Base CNN model	450
Pruned CNN Model	300
CNN Model from previous work	600



Model Size

- The .h5 file size of all the models are compared.

Model	Model Size (KB)
Base CNN model	238.98
Pruned CNN Model	64.3
Quantized Model	63
Pruned&Quantized	39.56
CNN Model from previous work	432



Android Application

- Started working on Android application once again.
- Finished basic app development tutorial on Kotlin.
- Successfully achieved the accelerometer and gyroscope readings from the phone.
- Currently working on the helper functions that will use the accelerometer and gyroscope values to achieve a prediction from the ML model.

Recent optimization techniques

- Read up on recent publications my MIT on AutoML and other papers published on AutoML and AutoKeras. . http://news.mit.edu/2020/foolproof-way-shrink-deep-learning-models-0430
- Got a bit off-track on these techniques but I got a good idea on the state-of-art optimization techniques present in this field.

Recent Paper on HAR with code(2019-2020)

- Jordao et al., “Human Activity Recognition Based on Wearable Sensor Data: A Standardization of the State-of-the-Art”. https://arxiv.org/pdf/1806.05226v3.pdf
- This paper implements and analyzes different sample generation process and validation protocols and gives a comparison between them.
- It gives a good comparison of different pre-processing techniques in the raw datasets
- Other papers are based on using computer vision for Activity Recognition

Recent Papers with code on model compression(2018-2020)

1. Gale et al., “The State of Sparsity in Deep Neural Networks” (https://paperswithcode.com/paper/the-state-of-sparsity-in-deep-neural-networks) (2019)

- This paper evaluates three state-of-the-art techniques to induce sparsity: Magnitude Pruning, Variational Dropout, l_0 regularization, and random pruning baseline.
- The dataset used to evaluate these techniques are Transformer trained on WMT 2014 English-to-German, and ResNet-50.

2. Polino et al.,” Model Compression via distillation and quantization” (https://paperswithcode.com/paper/model-compression-via-distillation-and) (2018)

- This paper proposes two compression methods, which combines both, weight quantization and distillation.
- The first method, quantized distillation, uses distillation during training process. The distillation loss is expressed with respect to the teacher network into the training of smaller student network whose weights are quantized.
- The second method, differentiable quantization, optimizes the location of quantization points through SGD.
- These methods are tested on smaller datasets like CIFAR-10 as well as larger datasets like Imagenet.

3. Fan et al., “ Training with Quantization Noise for Extreme Model Compression” (https://paperswithcode.com/paper/training-with-quantization-noise-for-extreme)(2020)

- This paper extends the quantization aware training approach. The authors propose to only quantize a different random subset of weights during each forward.
- The dataset used for experiments is Imagenet.

Papers read on Joint Pruning and Quantization

1. Utilizing Explainable AI for Quantization and Pruning of DNN. https://arxiv.org/pdf/2008.09072.pdf
   - Utilizes DeepLIFT algorithm for pruning and quantization of DNNs.
   - DeepLIFT compares activation of each neurons to its reference activation and assigns scores according to the difference.
   - Author presents a DeepLIFT-based layer sensitivity analysis. This method first calculates the number of neurons that need to be pruned in each layer. The number of neurons is calculated from the sensitivities of the layers. The sensitivity of layers is set by the optimization criteria (Multiply-Accumulate Operations or Number of Parameters). After this, the least important neurons in the layer are pruned.
   - The authors also propose a method for DeepLIFT-based quantization weight sharing and DeepLIFT based mixed-precision integer quantization.
   - The authors use Pytorch for deep-learning, pytorch-nemo for mixed-precision quantization-aware training, and pftops for profiling MACs and NPs

2. Structured Compression by Weight Encryption for Unstrucutred Pruning and Quantization. https://arxiv.org/pdf/1905.10138.pdf

- This paper presents a weight representation scheme for Spare QNN.
- Unstructured pruning method is used for weight representation. This representation is encrypted in a structured format, which is decoded using XOR-gate network during inference.
- An XOR-gate is used as a random number generator. The weights are quantized to binary weights. Encryption is done by reshaping the binary weight matrix as a 1D vector. Decryption is done through XOR-gate where encrypted weights vector is changed to its original bits consisting of care bits and rest is filled by randomly generated don’t care bits.
- The author proposes a design of an efficient XOR-gate for pruning and quantization of weight vectors.

3. Learning Sparse Low-Precision Neural Networks With Learnable Regularization. https://arxiv.org/pdf/1809.00095.pdf

- The authors propose a learnable regularization coefficient with MSQE regularizer for convergence of high-precision weights to their quantize values. For weight pruning, partial L2 regularization is used.
- A high-precision-model is retrained with partial L2 regularizer. Further, the model is retrained with MSQE regularizer and uniform weight quantization is performed on the model. A low-precision model with fixed-point weights is achieved and compression is done through entropy coding.
- The author proposes a method to reduce the accuracy loss caused by mismatch in the forward and backward passes. The proposed approximation gradually approaches to quantization function in training by adjusting the parameter in sigmoid function. Regularization is used for high-precision weight convergence which in turn reduces the mismatch between forward and backward pass.

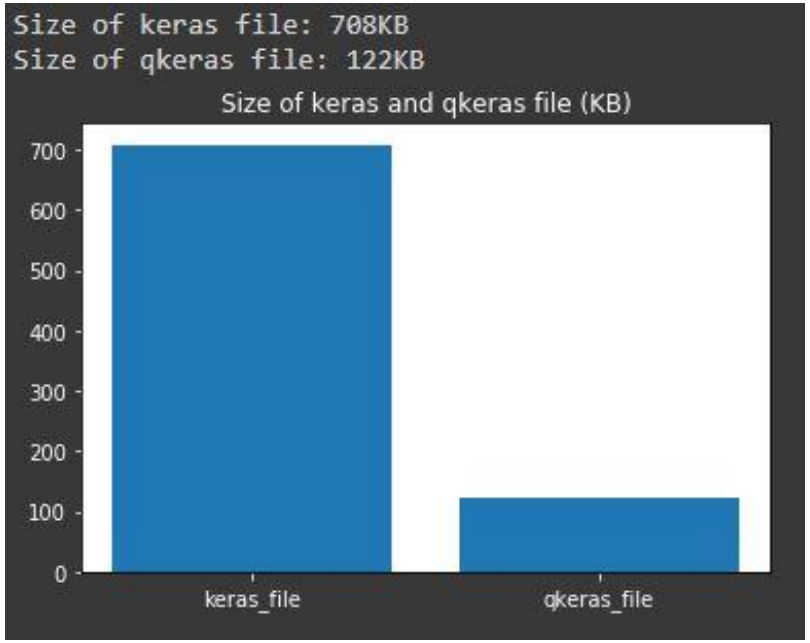
4. Differentiable Joint Pruning and Quantization for Hardware Efficiency. https://arxiv.org/pdf/2007.10463v1.pdf

- The authors propose a differentiable joint pruning and quantization scheme which uses structured pruning and mixed-bit precision quantization and incorporates into a single differentiable loss function.
- The authors show that on Bit-Operations, the complexity of the model is significantly reduced without degradation of accuracy.
- The proposed joint scheme is fully end-to-end and requires training only once.
- Quantization is performed through learnable mapping. A non-linear function is used to map any weight input. Structured pruning is done through variational information bottle-neck approach. Multiplicative Gaussian gates is added to all channels in a layer and a variational posterior Is learnt of weight distribution aiming at minimizing the mutual information between current and next layer's outputs. The learned distributions are then used to determine the layers to be pruned.
- A loss function incorporating the Bit-Operation count is used which allows joint optimization of pruning and quantization parameters.

Model Improvement

- Working on using Pytorch on HAR CNN models.
- Researching into pytorch-nemo for quantization aware training.
- Researching into Qkeras. Executed the Qkeras for CNN and RNN on CIFAR-10 dataset.
- Played around with the layers to see effect on model efficiency and file size.

- The file sizes of keras and qkeras model of CIFAR-10 dataset.



- Prediction Accuracy, Keras =98.84%

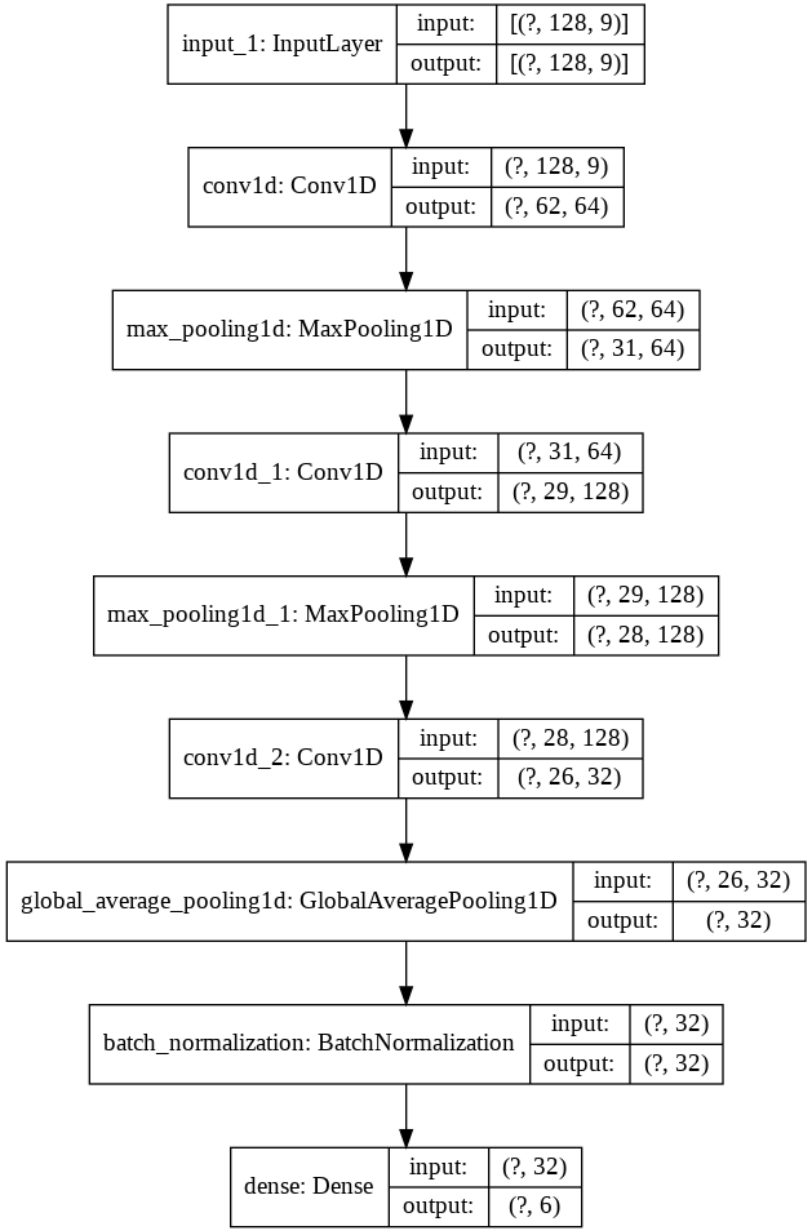
  Qkeras=98%

- Working on using this feature into HAR CNN models.

Papers read on Lightweight Model

- “LSTM-CNN Architecture for Human Activity Recognition” https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9043535
- “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design”. https://openaccess.thecvf.com/content_ECCV_2018/papers/Ningning_Light-weight_CNN_Architecture_ECCV_2018_paper.pdf
- “An Efficient and Lightweight Convolutional Neural Network for Remote Sensing Image Scene Classification.” https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7181261/
- “A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices.” https://arxiv.org/ftp/arxiv/papers/1909/1909.12917.pdf
- “An Energy-Efficient Method for Human Activity Recognition with Segment-Level Change Detection and Deep Learning”. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6749525/

- Researched into light-weight models especially for time-series datasets to have a small number of parameters with good accuracy.
- Changed base CNN model from sequential model to Keras Functional API model.
- Implemented a variety of neural network layers influenced from the above-mentioned papers.
- Observed the parameters and accuracy generated from the respective models.
- A promising result was shown in the following architecture.



CNN layer architecture

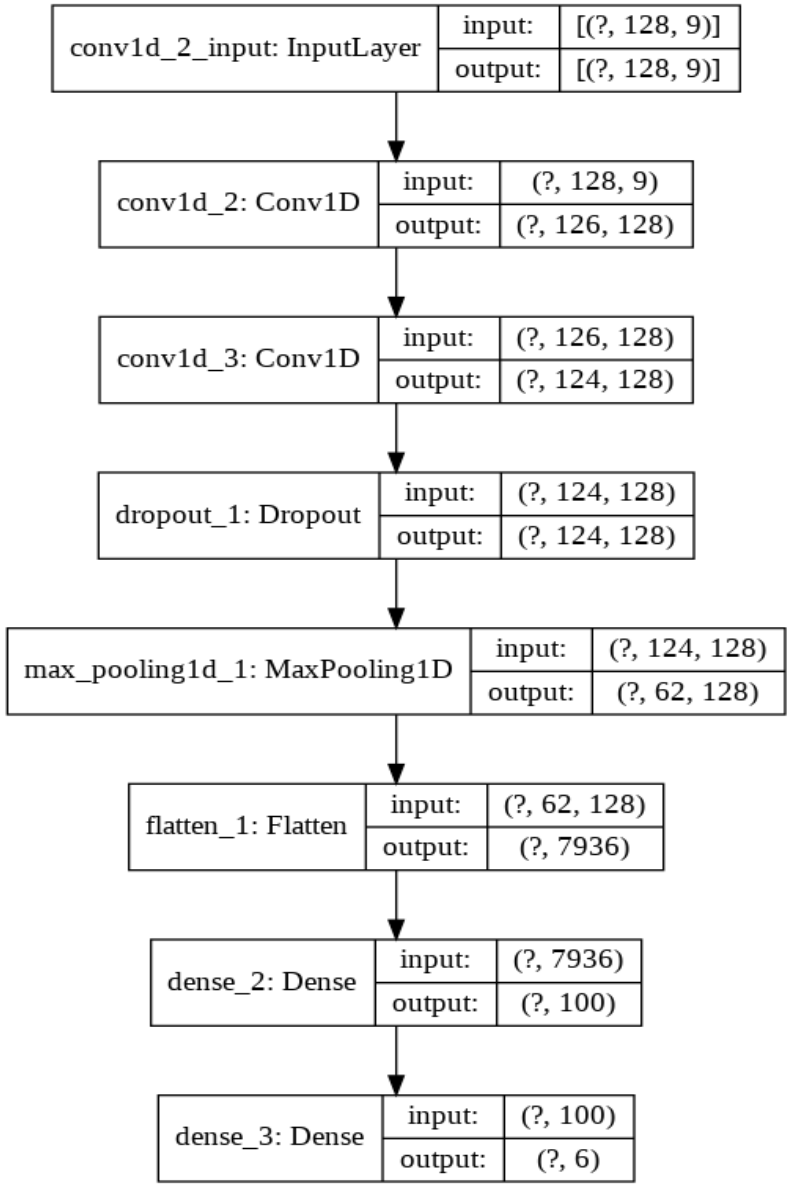
Model: "functional_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 9)]	0
conv1d (Conv1D)	(None, 62, 64)	2944
max_pooling1d (MaxPooling1D)	(None, 31, 64)	0
conv1d_1 (Conv1D)	(None, 29, 128)	24704
max_pooling1d_1 (MaxPooling1	(None, 28, 128)	0
conv1d_2 (Conv1D)	(None, 26, 32)	12320
global_average_pooling1d (Gl	(None, 32)	0
batch_normalization (BatchNo	(None, 32)	128
dense (Dense)	(None, 6)	198

Total params: 40,294
Trainable params: 40,230
Non-trainable params: 64

CNN layer parameters

Previously used Basic CNN Model Architecture




```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 126, 128)          3584
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 124, 128)          49280
_________________________________________________________________
dropout_1 (Dropout)          (None, 124, 128)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 62, 128)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7936)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               793700
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 606
=================================================================
Total params: 847,170
Trainable params: 847,170
Non-trainable params: 0
_________________________________________________________________
```

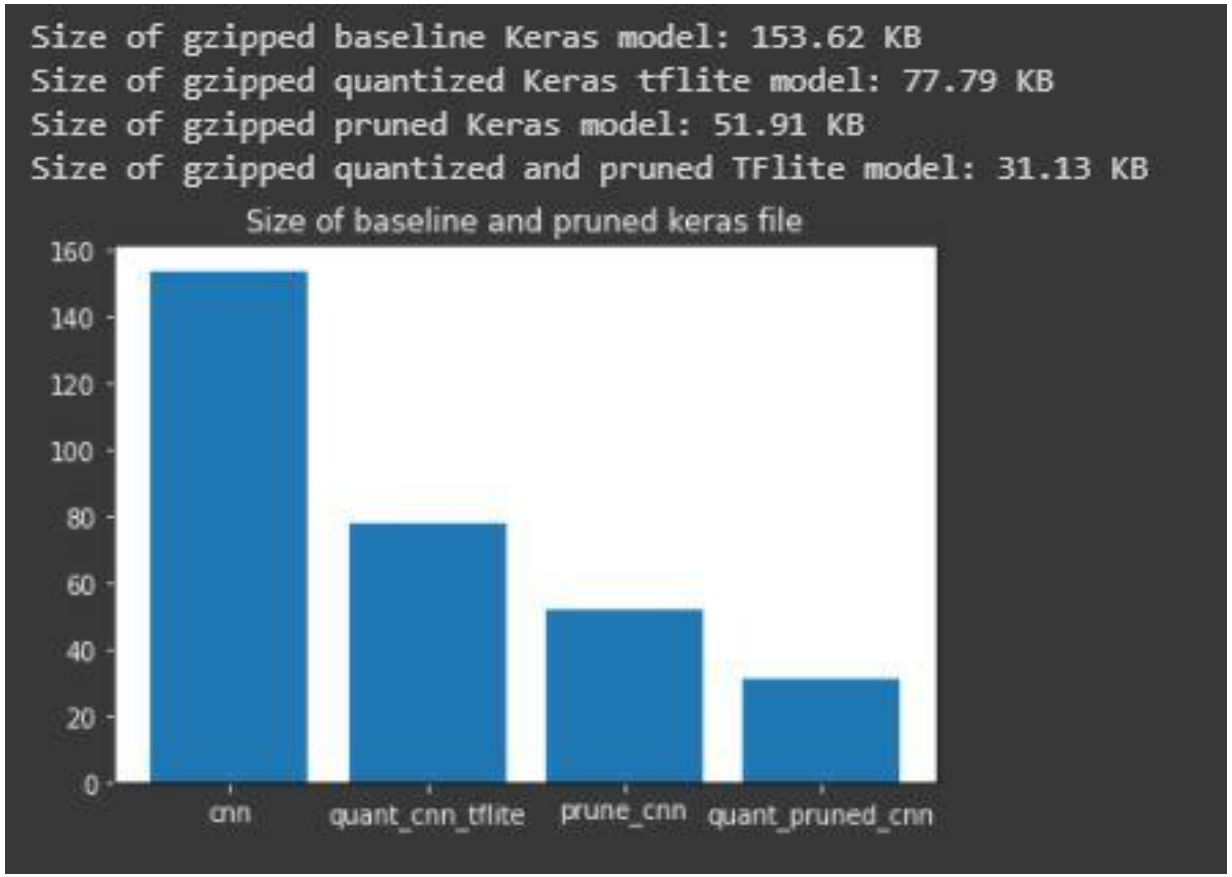
Results of new CNN model

- The accuracy of the proposed CNN model is observed. The model is then converted to tflite model using post-training quantization of float16. Weight Pruning is applied on the CNN model. The weight pruned model is then converted to tflite model. Accuracy of all the models is compared below.
- Accuracy result of base model, post-training quantized model, pruned model, and pruned and quantized model:

Model	Accuracy
CNN	95
Quantized CNN	87
Pruned CNN	94
Pruned & Quantized CNN	83

Results(Contd.)

- Model size of base model, post-training quantized model(tflite), pruned model, and pruned and quantized model(tflite):



Comparison with Previous Work (Parameter Count)

- Comparing parameter count of implemented CNN model with previous work
  - CNN model presented by Heeryon Cho and Sang Min Yoon in the paper " Divide and Conquer-based 1D CNN Human Activity Recognition Using Test Data Sharpening "
  - "Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors". https://github.com/guillaume-chevalier/HAR-stacked-residual-bidir-LSTMs

Model	Parameter Count
Implemented CNN	40,294
Divide&Conquer CNN	1,126,846
Bidir-LSTM-RNN	11,829

Comparison with Previous Work (Accuracy)

Model	Accuracy
Implemented CNN	95
Divide&Conquer CNN	96
Bidir-LSTM-RNN	92

Comparison with Previous Work (Model Size)

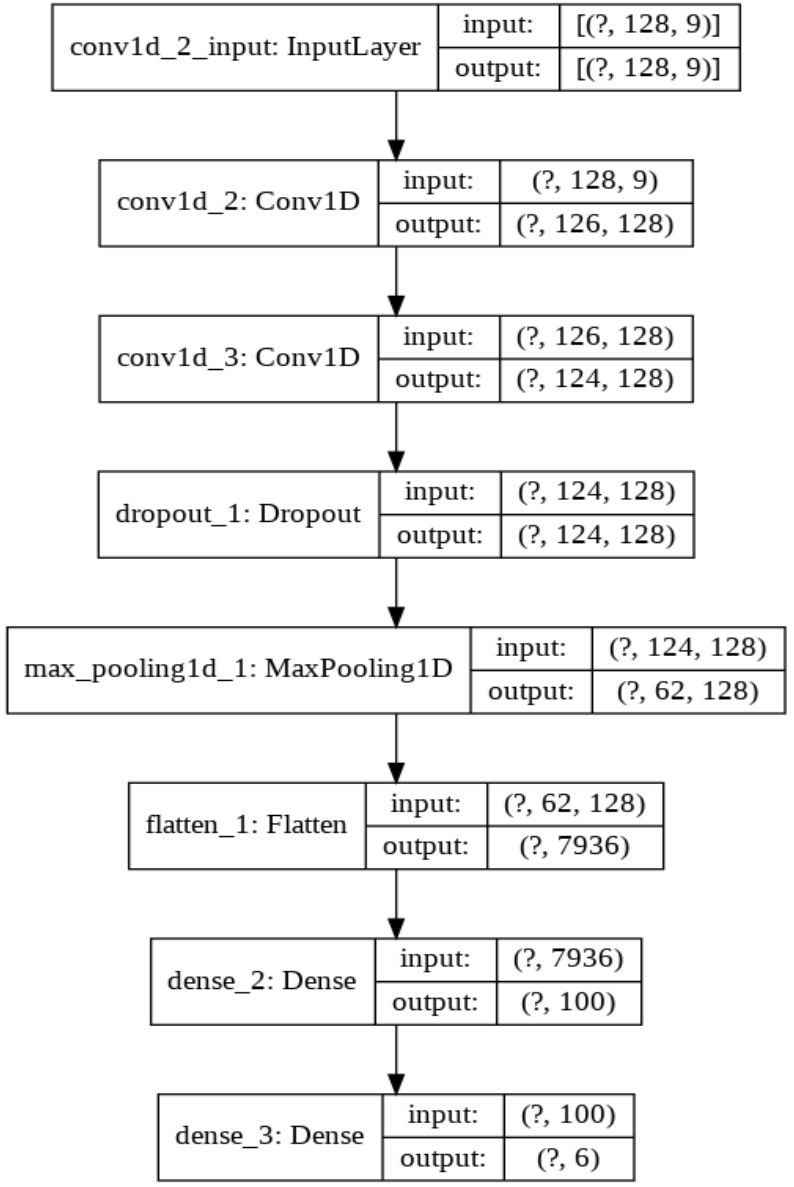
Model	Model Size (KB)
CNN	153
Divide & Conquer CNN	787
Bidir-LSTM-RNN	894

Ongoing & Future work

- Tried to implement Quantization Aware Training using tensorflow QAT but it does not support layer quantization of convolution 1D layer.
- Currently working on implementing QAT using QKERAS layers.
- Once I have implemented quantization aware training on the CNN layers, I will implement QAT with pruning and observe the results of accuracy and model size.
- Plan to add LSTM layer and observe the model properties of proposed model.
- Work on android application to deploy the models on an android phone to get real-time prediction and latency.

Implementation of Qkeras

- Researched into Qkeras for QAT.
- Currently present Qkeras layers: Qdense, QConv1D, Qactivation, QBatchNormalization.
- Executed the available layers in the CNN Model.
- Used Keras tutorial python notebook to apply Qkeras on CNN model.
- Ref: https://notebook.community/google/qkeras/notebook/QKerasTutorial

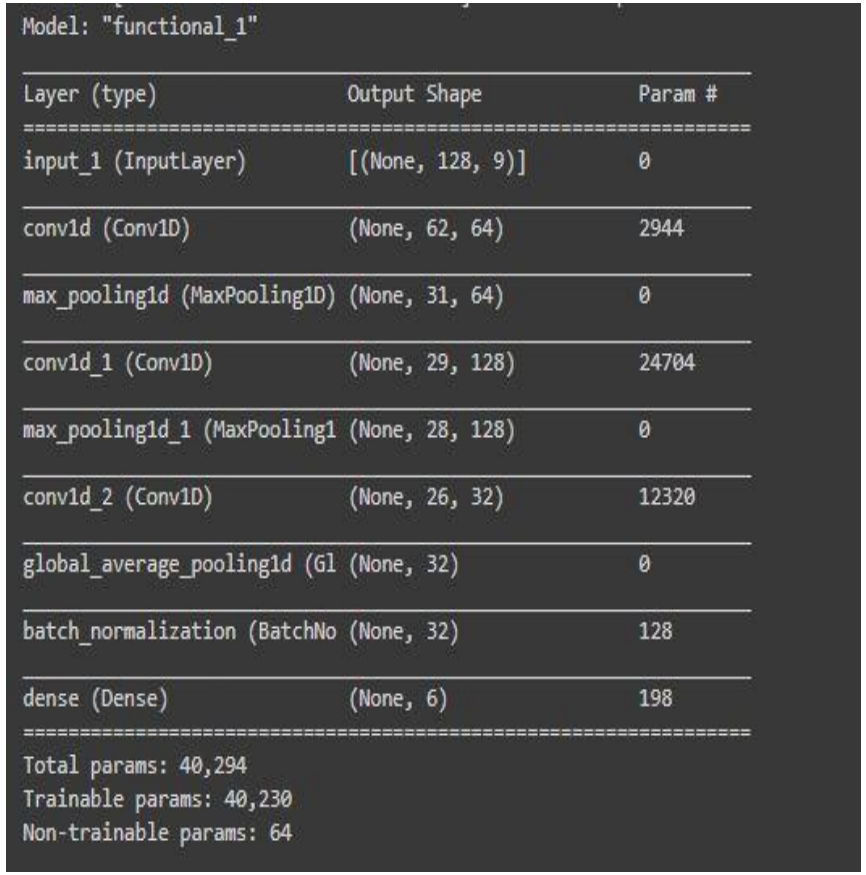


Result: Accuracy and Model Size

- Didn’t observe much change in the result.
- Accuracy of Qkeras and CNN Model are compared.
  - Accuracy of CNN model: 95.24%
  - Accuracy of Qkeras Model: 94.94%
- Model Size of Qkeras and CNN models are compared:
  - Size of CNN model: 153.55 KB
  - Size of Qkeras model: 153.83 KB
- Parameter Count of Qkeras and CNN models are compared:
  - Parameter Count of CNN model: 80,524
  - Parameter Count of Qkeras model: 80,524

Result: Parameters

- Comparing parameter count of CNN model and Qkeras Model:

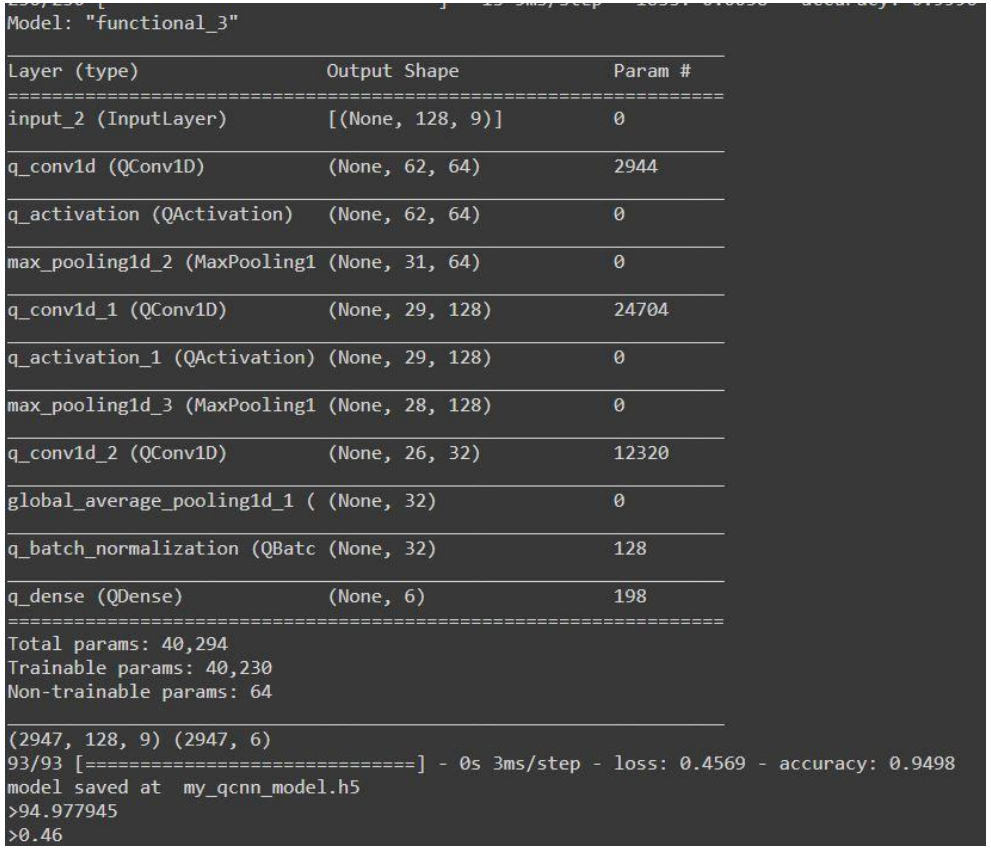


Model: "functional_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 9)]	0
conv1d (Conv1D)	(None, 62, 64)	2944
max_pooling1d (MaxPooling1D)	(None, 31, 64)	0
conv1d_1 (Conv1D)	(None, 29, 128)	24704
max_pooling1d_1 (MaxPooling1	(None, 28, 128)	0
conv1d_2 (Conv1D)	(None, 26, 32)	12320
global_average_pooling1d (Gl	(None, 32)	0
batch_normalization (BatchNo	(None, 32)	128
dense (Dense)	(None, 6)	198

Total params: 40,294
Trainable params: 40,230
Non-trainable params: 64

CNN Model



Model: "functional_3"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 128, 9)]	0
q_conv1d (QConv1D)	(None, 62, 64)	2944
q_activation (QActivation)	(None, 62, 64)	0
max_pooling1d_2 (MaxPooling1	(None, 31, 64)	0
q_conv1d_1 (QConv1D)	(None, 29, 128)	24704
q_activation_1 (QActivation)	(None, 29, 128)	0
max_pooling1d_3 (MaxPooling1	(None, 28, 128)	0
q_conv1d_2 (QConv1D)	(None, 26, 32)	12320
global_average_pooling1d_1 (	(None, 32)	0
q_batch_normalization (QBatc	(None, 32)	128
q_dense (QDense)	(None, 6)	198

Total params: 40,294
Trainable params: 40,230
Non-trainable params: 64

(2947, 128, 9) (2947, 6)
93/93 [==============================] - 0s 3ms/step - loss: 0.4569 - accuracy: 0.9498
model saved at my_qcnn_model.h5
>94.977945
>0.46

Qkeras CNN Model

Ongoing Work

- Just QAT using Qkeras does not seem to have any effect on the model.
- Planning to try a variation of QAT, Pruning, and Post-training quantization to see an effect on the accuracy and model size of CNN model. Mainly using pruning first, and then QAT, followed by tflite conversion.
- Started working on the android application for final deployment of compressed models.

Future Plan

Date	Work
10/10/20-10/14/2020	Work on implementing a few combinations of pruning& quantization and obtain results
10/15/2020-10/22/2020	Work on the android application to deploy the compressed CNN models
10/23/2020-10/30/2020	Finish working on the models and work on drafting the project report.
10/31/2020-11/6/2020	Work on report draft and submit final GS24 form on 11/6/2020
11/7/2020-12/15/2020	Make required changes in the project and prepare final report.

Qkeras Documentation

- Paper published by Google on Qkeras: "Ultra Low-Latency, Low-area Inference Accelerators using Heterogeneous Deep Quantization with Qkeras and hls4ml".
- Authors present a quantization-aware training for DNNs to generate low-latency inference IPs.
- Qkeras enables independent quantization of trainable parameters of layers. It provides mantissa quantization, exponent quantization, binary and ternary quantizers.
- In Qkeras, we need to specify quantizers for inputs, trainable parameters and output of the data manipulation layers that change data types. Thus we apply Qkeras to layers like Conv1D, Dense, but not to Flatten layer that only performs data transport.

Qkeras Documentation

- The following quantizers are present in Qkeras

Table 1: Quantizers

Quantizer
quantized_bits(bits, integer, symmetric, keep_negative, alpha, use_stochastic_rouding)
bernoulli(alpha, temperature, use_real_sigmoid)
binary(use_01, alpha, use_stochastic_rounding)
stochastic_binary(alpha, temperature, use_real_sigmoid)
ternary(alpha, threshold, use_stochastic_rounding)
stochastic_ternary(alpha, threshold, temperature, use_real_sigmoid)
quantized_relu(bits, integer, use_sigmoid, use_stochastic_rounding)
quantized_tanh(bits, integer, symmetric, use_stochastic_rounding)
quantized_po2(bits, max_value, use_stochastic_rounding, quadratic_approximation)
quantized_relu_po2(bits, max_value, use_stochastic_rounding, quadratic_approximation)

Qkeras Documentation

- *Quantize_bits* perform mantissa quantization

$$2^{\text{integer-bits+1}}\text{clip}(\text{round}(x*2^{\text{bits-integer-1}}),-2^{\text{bits-1}},2^{\text{bits-1}}-1),$$

- *Bernoulli* quantizer computes sigmoid(x) as the probablity measure that x is 1.
- *Binary* Quantizer generates either a -1 or +1. *Stochastic_binary* quantizer behaves the same as Bernoulli, just that it returns -1 or +1.
- *Ternary* quantizer generates -1,0, or +1. *Stochastic_ternary* adds regularization to the weights or input tensors.
- *Quantized_relu* and *quantized_tanh* performs quantized version of 'relu' and 'tanh'.
- *Quantized_po2* and *quantized_relu_po2* performs exponent quantization. Given an input as *(-1)^{s}2^{e}m (signed bits, exponent, significand)* , it returns the quantized number as

$$(-1)^s 2^{\text{quantized\_bits}(e+\text{round}(m))}$$

Qkeras Documentation

- *Po2* quantizer is efficient for multiplication as it does that in power-of-2 numbers that is achieved by performing the addition of exponents.
- The only disadvantage of *Po2* is that it doesn't have a representation of 0.
- Examples given by the qkeras implements
  - *po2* quantization on the convolution layer and activation layer of the CNN model, and *Quantized_bits* quantizer on the QDense layer while using regular Flatten layer and regular final softmax activation layer on the MNIST model.
  - batch normalization with po2 quantizer for gamma, variance, and beta quantizer. This layer is implemented after convolution layers which by itself uses ternary kernel quantizer on the MNIST model.
  - Pruning on a basic Qkeras models with convolution layers implementing quantized_bits quantizer. Magnitude pruning is applied on the model and conversion to tflite model is implemented on the MNIST model.

Ongoing Work

- Inspired from the MNIST examples, currently implementing such layers on the HAR CNN Model.
- As the papers mentions, Qkeras is implemented on layers that require trainable parameters and activation layers. Using this fact, the ongoing work is on implementing Qkeras on the required layers such as convolution, activation, and pooling layers.

Date	Work
10/16/20-10/21/2020	Execute different quantizers presented by Qkeras on HAR Model
10/22/2020-10/29/2020	Work on the android application to deploy the compressed CNN models
10/29/2020-11/6/2020	Work on report draft and submit final GS24 form on 11/6/2020
11/7/2020-12/15/2020	Make required changes in the project and prepare final report.

Qkeras on CNN Model

- The author of the Qkeras paper puts forth different quantizers that Qkeras function utilizes.
- The examples provided in their Github uses MNIST and CIFAR10 executes *Power of 2 quantizer, quantized_bits, and quantized_bits* in their CNN Model.
- A similar execution of these quantizers is done on the HAR CNN Model.
- A comparison is done between accuracy and model size of the original CNN model and the qkeras CNN model.

Qkeras on CNN Model

- Qkeras implements quantization aware training on the CNN model.
- Qkeras layers are mainly applied to data manipulation layers such as activation(Qactivation) and convolution(QConv1D) layers.
- The Qactivation layer utilizes *quantized_relu_po2* which performs exponent quantization.
- The QConv1D utilizes *quantized_po2* for kernel quantizer,and *quantized_bits* for bias_quantizer. This model is then converted to a tflite model.
- Furthermore, we perform magnitude pruning on the model to see its effect on the accuracy and model size.

Model Characteristics of Qkeras CNN

- Minimum and Maximum weights of each layer.

```
input                        -15.7022  15.1091
acti                           0.0000   1.0000
conv1d_0_m                    -7.6143   5.0205 ( -1.0000   1.0000) ( -0.2500   0.2500)
act0_m                         0.0000   1.7500
conv1d_1_m                    -8.9668   9.3760 ( -0.5000   0.5000) ( -0.2500   0.5000)
act1_m                         0.0000   1.7500
conv1d_2_m                    -9.1084   8.8887 ( -0.5000   0.5000) ( -0.5000   0.5000)
act2_m                         0.0000   1.7500
bn1                          -10.5015  23.2428 (  1.0000   2.0000) ( -0.0625   1.0000) (  0.0142   1.4193) (  0.0010   0.2500)
act3_m                         0.0000   1.7500
dense                        -12.9375  14.3235 ( -0.8750   0.8750) ( -0.8750   0.6250)
softmax                        0.0000   1.0000
```

- Test loss and accuracy:

```
Test score: 0.26870110630989075
Test accuracy: 0.9222938418388367
```

Model Characteristics of Qkeras CNN

- Qmodel Stats.
- The figure shows the model stats and weight profiling of each layer.
- The sparsity of the model is also presented, and the total sparsity is of 0.0079.

```
Number of operations in model:
    conv1d_0_m                    : 217728 (sadder_4_4)
    conv1d_1_m                    : 1499136 (sbarrel_4_3)
    conv1d_2_m                    : 344064 (sbarrel_4_3)
    dense                         : 192   (smult_4_3)

Number of operation types in model:
    sadder_4_4                    : 217728
    sbarrel_4_3                   : 1843200
    smult_4_3                     : 192

Weight profiling:
    conv1d_0_m_weights            : 1728  (4-bit unit)
    conv1d_0_m_bias               : 64    (4-bit unit)
    conv1d_1_m_weights            : 24576 (4-bit unit)
    conv1d_1_m_bias               : 128   (4-bit unit)
    conv1d_2_m_weights            : 12288 (4-bit unit)
    conv1d_2_m_bias               : 32    (4-bit unit)
    dense_weights                 : 192   (4-bit unit)
    dense_bias                    : 6     (4-bit unit)

Weight sparsity:
... quantizing model
    conv1d_0_m                    : 0.0340
    conv1d_1_m                    : 0.0051
    conv1d_2_m                    : 0.0024
    dense                         : 0.4646
    ----------------------------------------
    Total Sparsity                : 0.0079
```

Model Characteristics of Qkeras CNN

- Model Summary is shown.
- Model Size: 443.68KB
- Post-training Float-16 quantization model size: 33.22 KB
- TFLite Model Accuracy:

```
Accuracy after 1000 data : 0.909000
Accuracy after 2000 data : 0.908000
Base TFLite test_accuracy: 0.9205972175093315
```

- As observed, there is not much change in accuracy after compression.

[15]

input (InputLayer)	[(None, 128, 9)]	0
acti (QActivation)	(None, 128, 9)	0
conv1d_0_m (QConv1D)	(None, 126, 64)	1792
act0_m (QActivation)	(None, 126, 64)	0
max_pooling1d (MaxPooling1D)	(None, 63, 64)	0
conv1d_1_m (QConv1D)	(None, 61, 128)	24704
act1_m (QActivation)	(None, 61, 128)	0
max_pooling1d_1 (MaxPooling1	(None, 30, 128)	0
conv1d_2_m (QConv1D)	(None, 28, 32)	12320
act2_m (QActivation)	(None, 28, 32)	0
global_average_pooling1d (Gl	(None, 32)	0
bn1 (QBatchNormalization)	(None, 32)	128
act3_m (QActivation)	(None, 32)	0
flatten (Flatten)	(None, 32)	0
dense (QDense)	(None, 6)	198
softmax (Activation)	(None, 6)	0

Total params: 39,142
Trainable params: 39,078
Non-trainable params: 64

Model Characteristics of Pruned Qkeras CNN

- The complete model is pruned with initial sparsity of 0.1 and final sparsity of 0.9.
- The sparsity of each layer is presented. The convolution layers have 90% sparsity.

```
prune_low_magnitude_conv1d_0_m: (conv1d_0_m/kernel:0, 0.8998842592592593)
prune_low_magnitude_conv1d_1_m: (conv1d_1_m/kernel:0, 0.8999837239583334)
prune_low_magnitude_conv1d_2_m: (conv1d_2_m/kernel:0, 0.8999837239583334)
prune_low_magnitude_dense: (dense/kernel:0, 0.9010416666666666)
```

- Total weights in each layer

```
conv1d/kernel:0 -- Total:2880, Zeros: 90.00%
conv1d/bias:0 -- Total:64, Zeros: 0.00%
conv1d_1/kernel:0 -- Total:24576, Zeros: 90.00%
conv1d_1/bias:0 -- Total:128, Zeros: 0.00%
conv1d_2/kernel:0 -- Total:12288, Zeros: 90.00%
conv1d_2/bias:0 -- Total:32, Zeros: 0.00%
batch_normalization/gamma:0 -- Total:32, Zeros: 0.00%
batch_normalization/beta:0 -- Total:32, Zeros: 0.00%
batch_normalization/moving_mean:0 -- Total:32, Zeros: 0.00%
batch_normalization/moving_variance:0 -- Total:32, Zeros: 0.00%
dense/kernel:0 -- Total:192, Zeros: 90.10%
dense/bias:0 -- Total:6, Zeros: 0.00%
```

Model Characteristics of Pruned Qkeras CNN

- Model Summary:
- Pruned model accuracy:

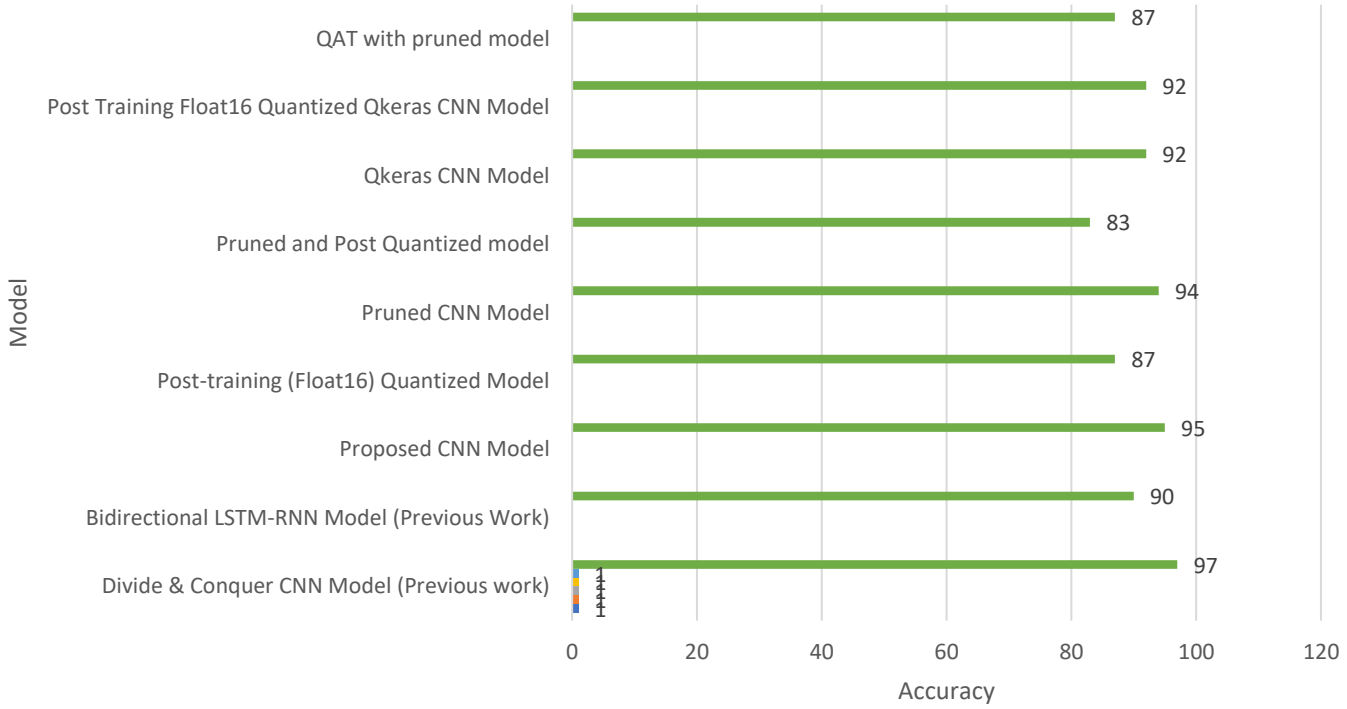
```
Test score: 0.499791562557220 46
Test accuracy: 0.8686800003051758
```

- Pruned model size: 298.67KB

```
_________________________________________________________________
input (InputLayer)           [(None, 128, 9)]          0
_________________________________________________________________
prune_low_magnitude_acti (Pr (None, 128, 9)            1
_________________________________________________________________
prune_low_magnitude_conv1d_0 (None, 126, 64)           3458
_________________________________________________________________
prune_low_magnitude_act0_m ( (None, 126, 64)           1
_________________________________________________________________
prune_low_magnitude_max_pool (None, 63, 64)            1
_________________________________________________________________
prune_low_magnitude_conv1d_1 (None, 61, 128)           49154
_________________________________________________________________
prune_low_magnitude_act1_m ( (None, 61, 128)           1
_________________________________________________________________
prune_low_magnitude_max_pool (None, 30, 128)           1
_________________________________________________________________
prune_low_magnitude_conv1d_2 (None, 28, 32)            24578
_________________________________________________________________
prune_low_magnitude_act2_m ( (None, 28, 32)            1
_________________________________________________________________
prune_low_magnitude_global_a (None, 32)                1
_________________________________________________________________
prune_low_magnitude_bn1 (Pru (None, 32)                129
_________________________________________________________________
prune_low_magnitude_act3_m ( (None, 32)                1
_________________________________________________________________
prune_low_magnitude_flatten_ (None, 32)                1
_________________________________________________________________
prune_low_magnitude_dense (P (None, 6)                 392
_________________________________________________________________
prune_low_magnitude_softmax  (None, 6)                 1
=================================================================
Total params: 77,721
Trainable params: 38,854
Non-trainable params: 38,867
```

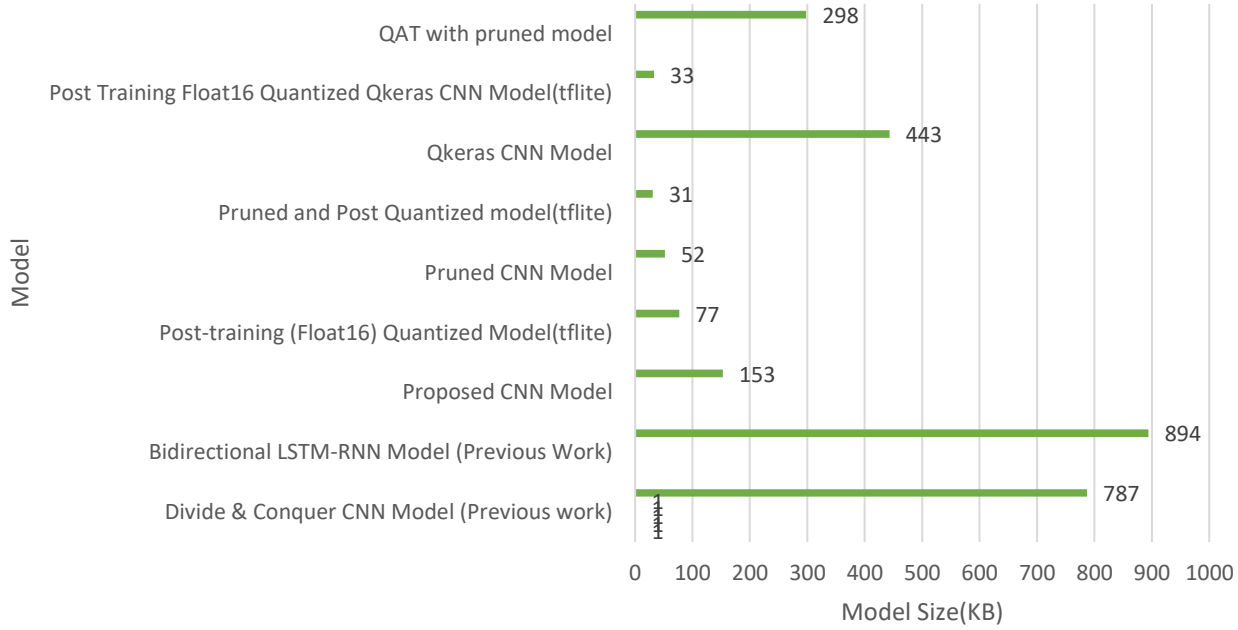
Comparison of all models: Accuracy

Model	Accuracy
Divide & Conquer CNN Model (Previous work)	97
Bidirectional LSTM-RNN Model (Previous Work)	90
Proposed CNN Model	95
Post-training (Float16) Quantized Model	87
Pruned CNN Model	94
Pruned and Post Quantized model	83
Qkeras CNN Model	92
Post Training Float16 Quantized Qkeras CNN Model	92
QAT with pruned model	87

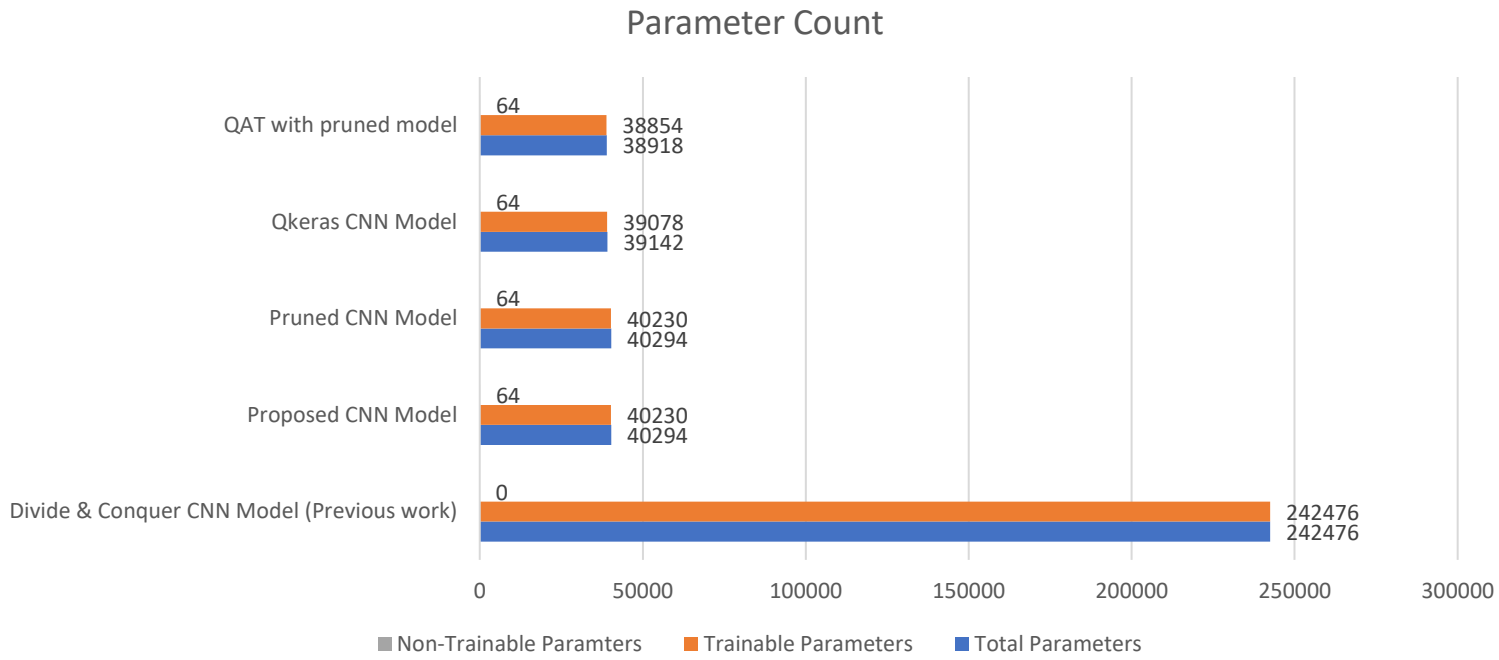


Comparison of all models: Model Size

Model	Model Size(KB)
Divide & Conquer CNN Model (Previous work)	787
Bidirectional LSTM-RNN Model (Previous Work)	894
Proposed CNN Model	153
Post-training (Float16) Quantized Model(tflite)	77
Pruned CNN Model	52
Pruned and Post Quantized model(tflite)	31
Qkeras CNN Model	443
Post Training Float16 Quantized Qkeras CNN Model(tflite)	33
QAT with pruned model	298



Comparison of all models: Parameter Count



Model	Total Parameters	Trainable Parameters	Non-Trainable Parameters
Divide & Conquer CNN Model (Previous work)	242476	242476	0
Proposed CNN Model	40294	40230	64
Pruned CNN Model	40294	40230	64
Qkeras CNN Model	39142	39078	64
QAT with pruned model	38918	38854	64

Conclusion

- A variety of quantization and pruning is performed on the proposed CNN Model such as:
  - Post training Quantization(Float16)
  - Magnitude Pruning of CNN Model
  - Post training quantization on pruned CNN Model
  - Quantization aware training using Qkeras on CNN Model
  - Post Training quantization on QAT CNN Model
  - Magnitude Pruning of Qkeras CNN Model
- The accuracy, model size, and parameter count of all the above models were compared. The best model accuracy trade-off with its model size is of Post training quantized Qkeras CNN Model with accuracy of 92% and model size of 32KB.

Ongoing Work

- Looking into an android application suitable to intake the tflite files of different CNN Model.
- Qkeras also has Qtools function which can be used for data type map generation and energy consumption estimation.
- Working on executing the above tool by Qkeras on CNN Models to get more insight on energy consumption of model.

Date	Work
10/22/2020-10/29/2020	Work on the android application to deploy the compressed CNN models & QTools
10/29/2020-11/6/2020	Work on report draft and submit final GS24 form on 11/6/2020
11/7/2020-12/15/2020	Make required changes in the project and prepare final report.

Custom Quantization/Pruning Methods

- Researched into recent papers on custom quantization and pruning.
- The following papers were good resources along with their Github links:
  - "MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization", Chen et al.(Github)
  - "Lookahead: A Far-sighte Alternated of Magnitude-based Pruning", Park et al. (Github)
  - "Dynamic Model Pruning with Feedback", Lin et al. (Github)

MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

- The author proposes a neural network for learning zero gradient (g_r).
- Previous papers propose learning quantized deep neural network to reduce model size and energy consumption which is done by converting full-precision weights (*r*'s) into discrete values (*q*'s) in a supervised training manner.
- However, the training process for quantization is non-differentiable which in turn leads to infinite or zero gradients (g_r). Most training-based quantization methods use Straight-Through-Estimator (STE) to approximate g_r.
- Thus the author proposes a neural network to learn zero gradient.

MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

- A meta network is proposed which is trained using gradient w.r.t discrete values (g_q) and full-precision weights (r) as inputs. And output g_r for weight updates.
- This method gives a solution to the non-differentiability.
- Existing training-based quantization are divided into two categories: STE and Non-STE methods. STE methods contain the non-differentiable discrete quantization function, whereas non-STE method refers to learning without STE by directly working on full-precision weights with a regularizer to obtain feasible quantization.

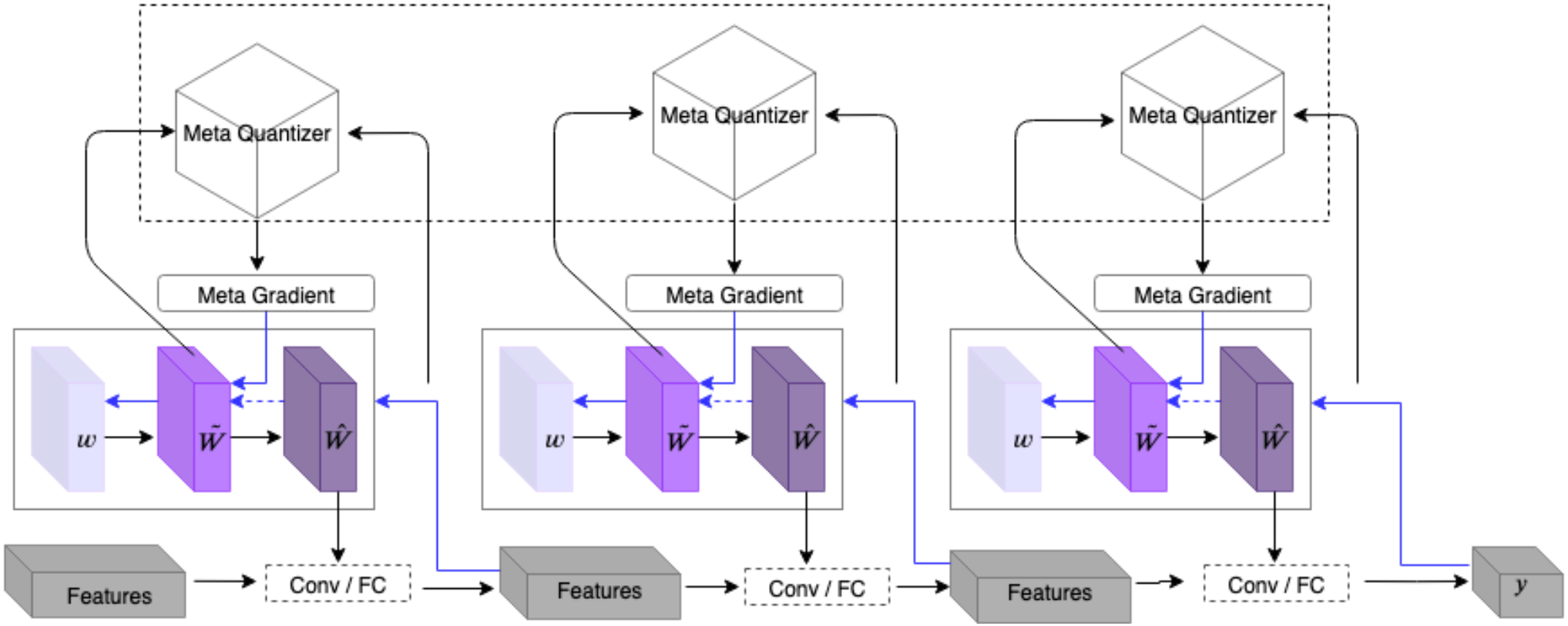
MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

- STE is widely used as it provides an approximated gradient for penetration of quantization function, but it brings the problem of gradient mismatch. This mismatch occurs as the gradient of the weights is not calculated using the values of the weights but rather its quantized value.
- To overcome the problem of gradient mismatch, the author proposes to learn $\frac{\partial Q(W)}{\partial W}$ by neural network ($\mathcal{M}$) in quantization training, where ***W*** is the full-precision weights and ***Q(·)*** is the quantization function. This is known as the *meta-quantizer* and is trained with the base quantized model.

MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

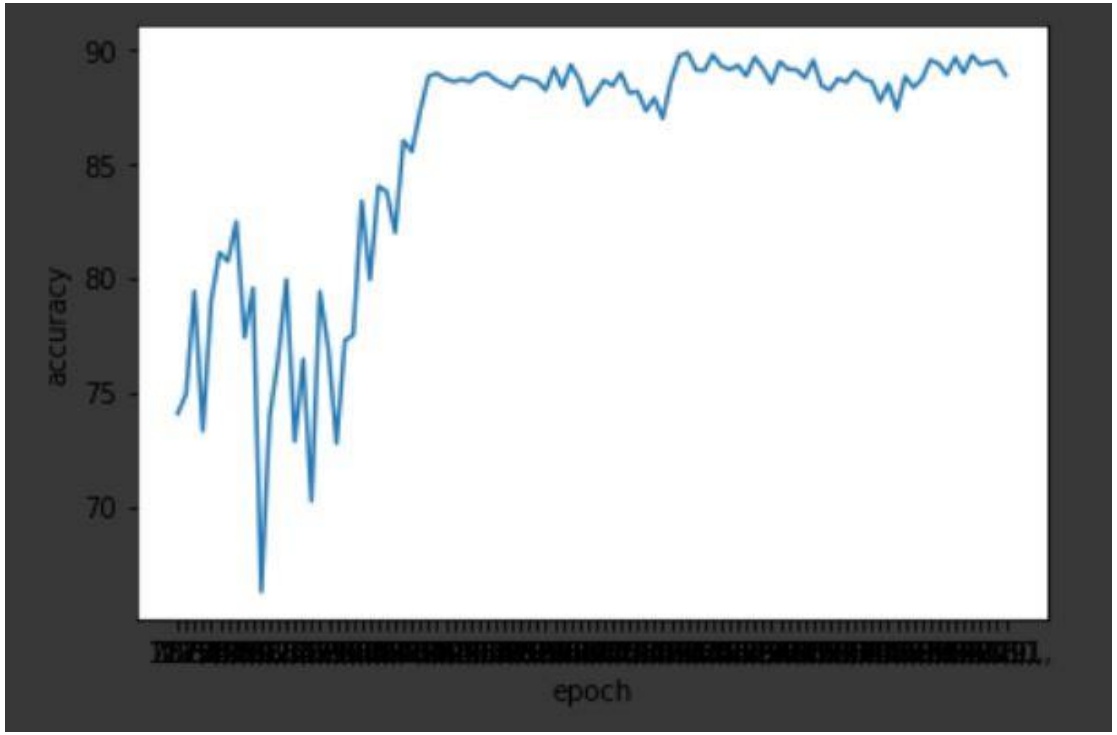
- For each backward propagation, $\mathcal{M}$ takes $\frac{\partial l}{\partial Q(W)}$ and $\boldsymbol{W}$ as inputs and the output is used to computer $\frac{\partial l}{\partial W}$ for updating weights using SGD or Adam optimizer.
- During this process, from the quantized weights to the full-precision weights, the gradient propagation is handled by $\mathcal{M}$. This overcomes the issue of non-differentiability and gradient mismatch.
- The gradients generated by the meta-quantizer are loss-aware.
- After quantization training is finished, the meta-quantizer can be removed and thus this method does not consume extra space for inference.

MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization



MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

- The code provided in the Github link was executed.
- Best Accuracy Model- ResNet20, Forward-Dorefa, Backward- MultiFC, Optimizer-SGD.
- Test accuracy – 89.820
- Test loss-0.16



MetaQuant: Learning to Quantize by Learning to Penetrate Non-differentiable Quantization

- Observation:
  - The paper proposes a unique way to quantize weights for quantization aware training.
  - The proposed neural network for quantizing weights could mean higher resource consumption while training the model. With the possibility of removing the meta-quantizer before deploying the model to resource constrained platform, this could mean higher convergence rate in the accuracy of the model while being used on edge devices.
  - The algorithm is tested on dorefa and BWN quantization functions and SGD and Adam optimization techniques. Where SGD gives the best performance.
  - The proposed method is implemented on image dataset of CIFAR10/CIFAR100. Future work will be done on implementing this method on time series dataset of Human Activity Recognition.

Ongoing Work

- Working on understanding the Meta-Quant paper to try to implement it on HAR dataset.
- Currently debugging the code of the following paper given on Github. Lookahead: A Far-sighte Alternated of Magnitude-based Pruning", Park et al. (Github)

Updated Slides

Dynamic Model Pruning with Feedback

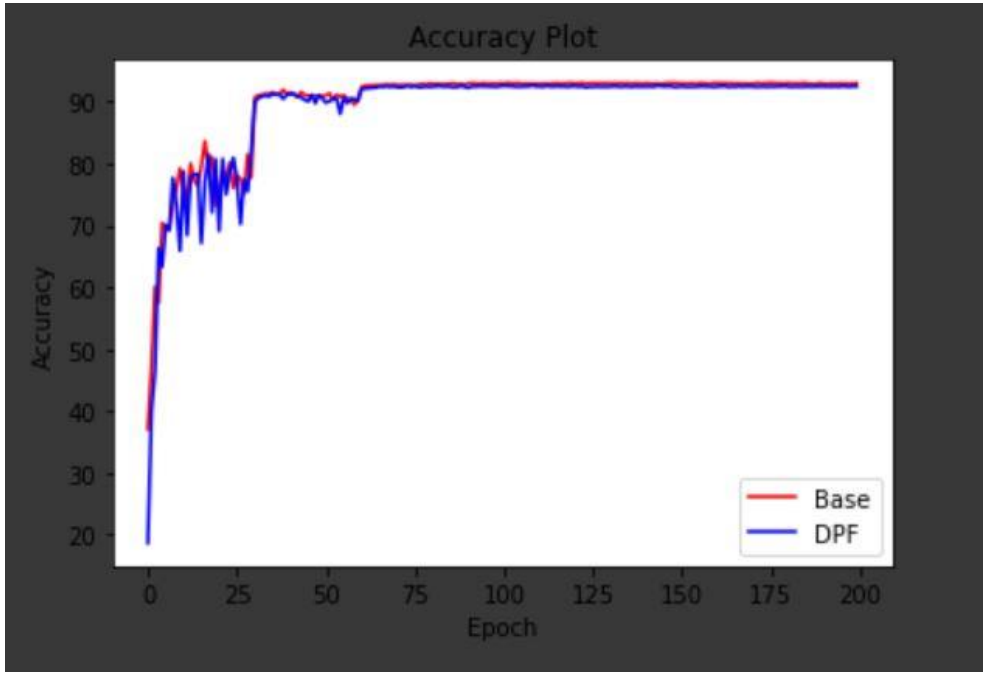
- This paper proposes a model compression method that generates a sparse trained model.
- The algorithm allows dynamic allocation of the sparsity pattern.
- A feedback signal is used to reactive prematurely pruned weight in one single training pass.
- The compression scheme uses a criterion that identifies important weights in the network throughout training to obtain appropriate masks.

Dynamic Model Pruning with Feedback

- The authors propose dynamic pruning with feedback (DPF) which evaluates the stochastic gradient of the pruned model and applies it to the dense model.
- Using the feedback allows to recover the important weights that were masked prematurely.
- The DPF scheme is evaluated on the CIFAR-10 dataset with ResNet model.
- DPF performs unstructured magnitude pruning across all the neural network layers.
- DPF is applied to all convolutional layers while keeping the last FC layer, biases and batch-normalization layer dense.
- The proposed algorithm has an increasing sparsity starting from 0 going to user defined limit.

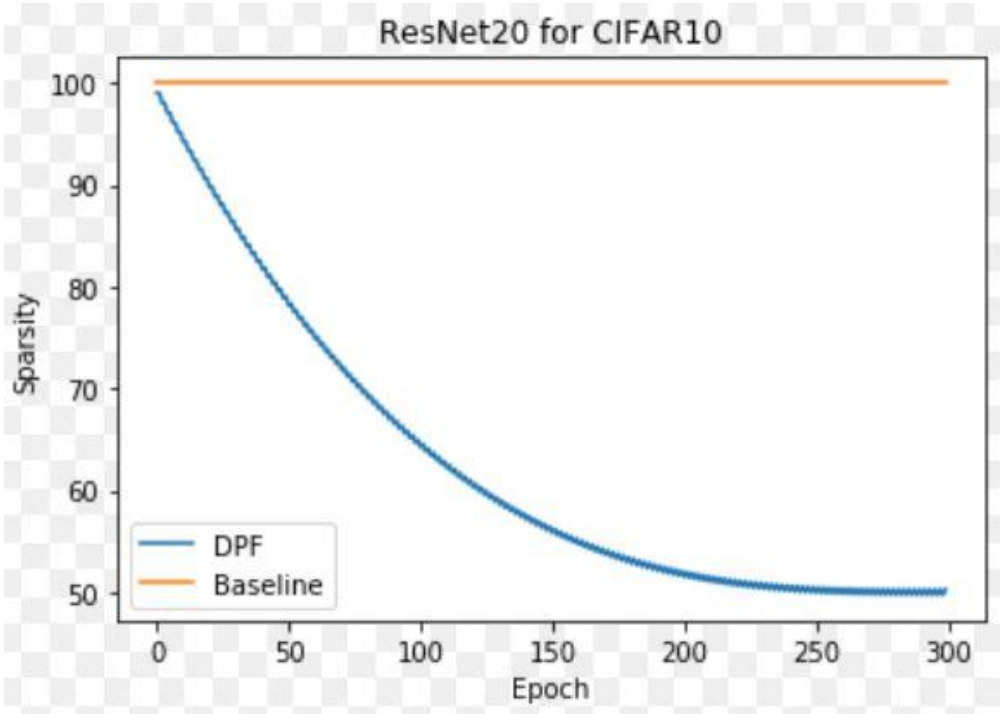
Dynamic Model Pruning with Feedback: Results

- The accuracy of baseline model with no pruning with dynamic pruning with feedback is shown below:
  - Base accuracy- 93.1%
  - DPF accuracy- 92.66%



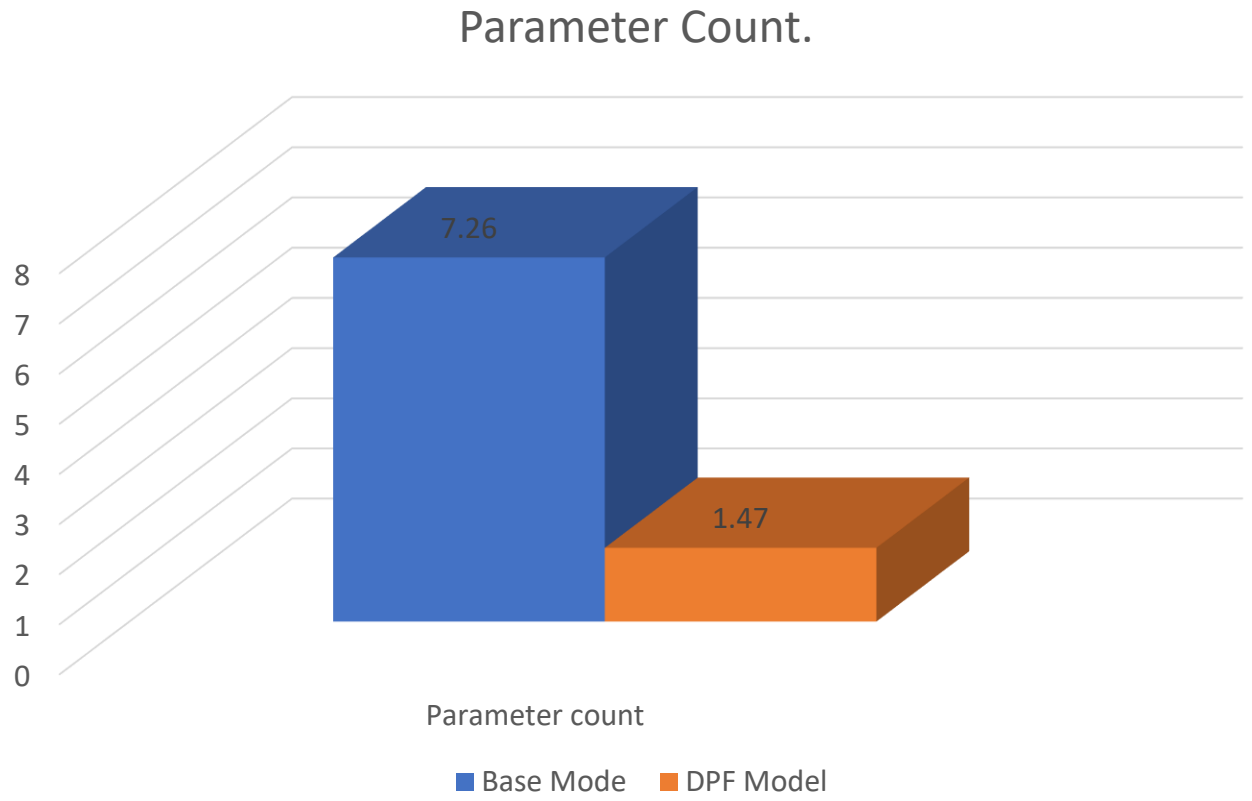
Dynamic Model Pruning with Feedback: Results

- The sparsity is varied in the DPF method. The graph below shows the same



Dynamic Model Pruning with Feedback: Results

- The original base model has 1.47M parameters whereas the DPF model has final parameter count of 6.79M.



Observation & Conclusion

- The algorithm shows promising results on dynamic pruning of CIFAR-10 model. It shows to retain accuracy while pruning down the parameters using dynamic feedback method.
- This algorithm could be used on time series datasets as it could show promising results of pruning on the model making it able to use on edge devices.
- The dynamic feedback on pruning is what attracts this method to be implemented on the time series datasets. As compared to the previous paper (MetQuant), which was based on a neural network to realize important weights that needed to be quantized.

Ongoing work

- Studying the code implemented by the authors to execute it on HAR dataset.
- Changing the given program to make it work on a time series dataset.