International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

# Optimizing Neural Networks for Embedded Systems

**Ranjith M S[1], Dr. S Parameshwara[2]**

[1]*Student, Dept. of ECE, National Institute of Engineering, Mysuru, India*
[2]*Associate Professor, Dept. of ECE, National Institute of Engineering, Mysuru, India*

---

**Abstract -** *Deep learning has dramatically increased the state of the art in Speech, Vision and many other areas. In fact, it is being applied in most of the fields from bio-medical to robotics. And all these deep learning models to perform inference needs lot of resources like large computation power since the device has to perform capturing data pre-process it then feed to neural network which has to perform large number of floating point operations varying from 3000 floating point operations to billions of floating point operations depending on the complexity of the neural network. Deploying such a system with deep neural network in cloud is highly limited as more latency is introduced since there is a round trip to server, privacy risks as the data needs to leave the device, connectivity as the data need to be sent to server, power consumption as network connections are power hungry. All these problems could be overcome by deploying the deep learning model in the edge devices like microcontroller, mobile devices. We achieve this by quantizing the trained neural network which reduces the size as well as increase the speed at the same time, the performance is not compensated.*

***Key Words***: Deep learning, Quantization, Microcontroller, Neural Networks, TensorFlow Lite, Embedded System, ARM Processor, Artificial Intelligence.

## 1. INTRODUCTION

Currently, a variety of embedded system are deployed but the usage of neural networks is limited when it comes to embedded systems like microcontrollers. Household devices like Refrigerators, washing machines uses set of logic, rules for their automatic operations. By optimizing the network trained on a dataset traditional way of controlling can be replaced by intelligently monitoring the systems with the power of AI and neural networks.

Deep learning is not only limited to Image data and Sequence predictions like in case of speech recognition. The deep learning technique is less explored on tabular data, but these ideas can be applied on tabular data. *Artificial Neural Networks Applied to Taxi Destination Prediction* paper propose to use DNN on tabular data "*we used an almost fully automated approach based on neural networks and we ranked first out of 381 teams*" [1]. Since we are using only dense networks we use *He initialization* [2] as compared with *Xavier Initialization* [3] for initializing the weights while training the network.

*""overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks"* [4].

The optimizers such as Adam [5], rms [6], LARS [7] leads for the faster convergence. "*One of the key elements of super-convergence is training with one learning rate cycle and a large maximum learning rate*" [8]. To speed up training we use *Batch Normalization* [9].

We train the neural network to identify the gestures based on sensor inputs, combination of accelerometer and gyroscope found to work well for training as well as gives good accuracy at test time. The trained network is then optimized by reducing the redundant computations which do not contribute much to the model accuracy also the pruning is done to reduce the floating-point precision. All these redundant computation and reduction in floating point precision is done based on the representative dataset which is similar to that of data given to the model at the test time so that the desired accuracy is maintained even after optimization.

This optimized trained neural network is converted into Flat buffer and this Flat buffer is converted into C array which can be integrated as header file and can be used to perform the classification task on the data collected from the sensor on to the board.

## 2. EXPERIMENTAL SETUP

We deploy the neural network on the edge device itself hence the data is read from the interfaced sensor using edge device, in this case we use Arduino UNO to read the data from MPU6050. MPU6050 is used for both dataset preparation and during the inference time. MPU6050 IMU has both 3-Axis accelerometer and 3-Axis gyroscope integrated on a single chip. It uses MEMS technology and the Coriolis Effect for measuring.

The dataset is trained with neural network on Jupyter Notebook IDE in Tesla K80 GPU. TensorFlow is used for implementation of neural network. TensorFlow Lite for quantizing the network.

### 2.1 Preparation of dataset

We combine the accelerometer and gyroscope to get very accurate information about the sensor orientation. Accelerometer and gyroscope values are read from MPU6050 using I2C communication protocol in Arduino Uno and this read data is sent serially to print on the serial monitor. MPU6050 is calibrated properly as it has the inbuilt errors.

We take 30 samples of data to identify each gesture, we set some threshold on the measured data to distinguish between the performance of gesture and still position. Once the action of gesture starts the threshold is met and the values are printed with each value separated by comma and each sample is printed in new line this helps in conversion of data to CSV format which can be used for neural network training. Each gesture is performed several times and the printed values are copied to CSV file and each gesture are saved in different CSV file with gesture name.

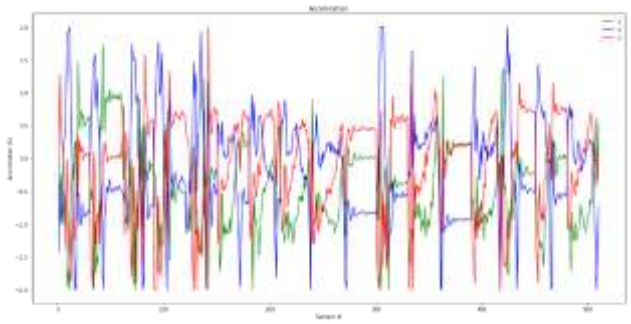

**fig -1**: accelerometer data

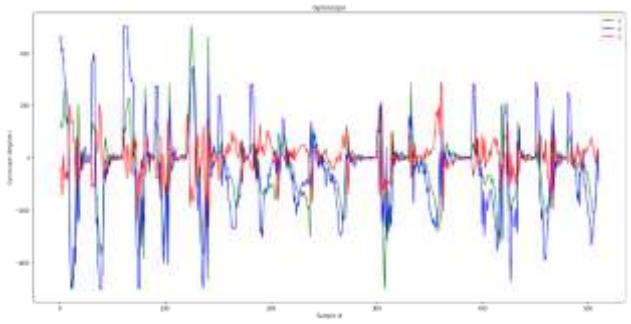

**fig -2**: gyroscope data

fig-1 represents how the value of acceleration varies in each sample while performing a particular gesture similarly fig-2 represents gyroscope data.

The CSV files are read as pandas DataFrame in python and normalization is done on the input data. The accelerometer values vary from -4 to +4 and gyroscope value varies from -2000 to +2000 we normalize these values to vary between 0 and 1 in order to make the training faster and to counter the problem of overshooting during optimization. Later normalized gesture data is saved in the form of array with shuffling the order of data and corresponding label's one hot encoding is stored in another array. Shuffling is done in order to avoid problems like catastrophic forgetting.

### 2.2 Training the Neural Network

TensorFlow keras is used for implementation and training of the neural network. Since 30 samples are captured for each gesture performed we have 180 values for each gesture as one sample contains 6 values where gyroscope measures rate of change of the angular position over time, along the X, Y and Z axis and accelerometer measures gravitational acceleration along the 3 axes. All these 180 inputs are fed into dense neural network with first hidden layer having 50 hidden units and ReLU as the activation function and the second layer has 15 hidden units and ReLU as activation the final output layer has hidden units based on number of unique gestures.

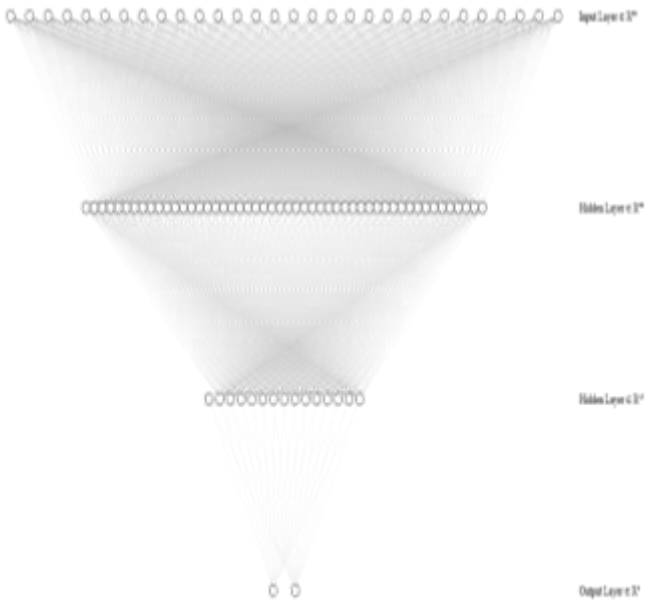

**fig -3**: Neural network architecture

**Table -1:** Network structure

| **Model structure and parameters** | | |
|---|---|---|
| **Layer** | **Output shape** | **NO. of parameters** |
| Dense_1 | (None, 1, 50) | 9050 |
| Dense_2 | (None, 1, 15) | 765 |
| Output | (None, 1, 2) | 32 |
| Total params: 9,847<br>Trainable params: 9,847<br>Non-trainable params: 0 | | |

The model was trained using 'rmsprop' optimizer for 600 epochs by which the validation loss was saturated to very low value. Categorical cross entropy is used as a loss function and the network is optimized based on that categorical cross entropy is SoftMax activation plus a cross entropy loss.

SoftMax function is given by,

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where C represents total number of class and $s_j$ are the score inferred by the neural network for each class in C.

Hence categorical cross entropy is given by,

$$CE = -\sum_i^C t_i log(f(s)_i)$$

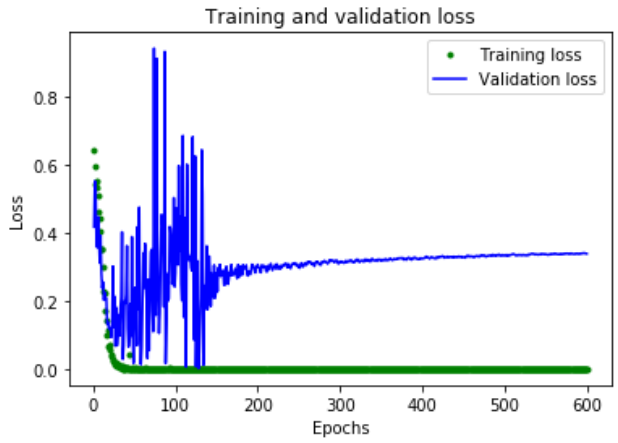


**fig -4**: Variation of loss during training

fig-4 shows how the loss varied with epochs during training and we observe that both training and validation loss reach saturation by 200 epochs.

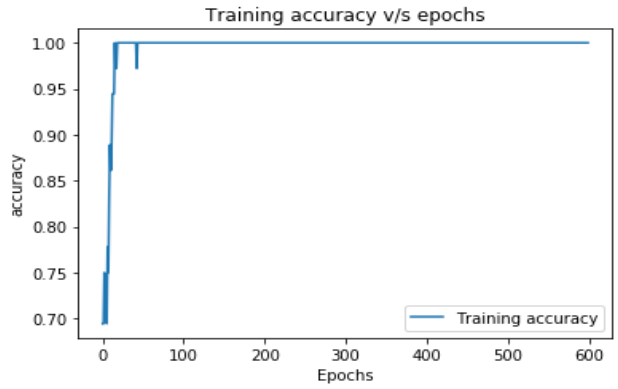


**fig -5**: Variation of accuracy during training

fig-5 shows how accuracy varied during training and we observe that the accuracy reaches its saturation by 100 epochs.

**Table -2:** Network results after training

| **Final loss and accuracy** | | |
|---|---|---|
| | accuracy | loss |
| Training | 1.0000 | 0.0000e+00 |
| validation | 0.9167 | 0.3393 |

From table-2 we see that the neural network has very low validation loss and high training and validation accuracy implying that the network has learnt to perform the task well.

The model was trained on Tesla K80 GPU which can do up to 8.73 Teraflops single-precision and up to 2.91 Teraflops double-precision performance with NVIDIA GPU Boost. This network gives a validation accuracy of 91.67 %.

### 3. QUANTIZATION

The trained model requires very high memory and computation power. This is quantized in order to reduce memory and computation power using TensorFlow lite.

The trained model is of 113 kilo bytes which is very hard to fit into microcontrollers as ARM cortex M3 generally have 64 Kilo bytes of RAM. So, this model is quantized by reducing the floating-point precision of all the weights of the neural network from 64 bits to 16/32 bits.

Pruning of parameters is done properly using the representative dataset during the model optimization i.e., reduce parameter count by structured pruning.

TensorFlow lite converts model to Flat buffer format(.tflite), the flat buffer is converted into C byte array in the form of simple C file. The converted ".h" format file could be used in embedded c program. ".tflite" model can be deployed on Raspberry pi as well. This model can use the additional computational resources like Neural compute stick if present in Raspberry pi.

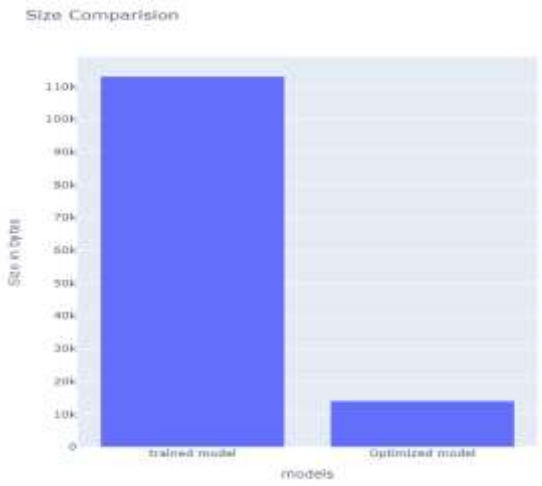


**fig -6**: Reduction in size

After Quantization the model size reduced to 14008 bytes which is approximately 13% of the initial size. Comparison of sizes is shown in fig-6. The size could be still reduced if the floating-point weights are converted to integer, but the accuracy of the model becomes less in that case.

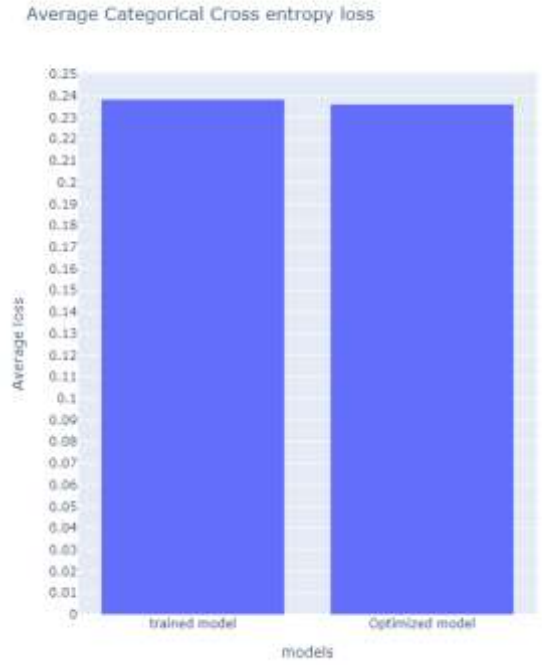


**fig -7**: avg. Categorical cross entropy loss

In rare cases, certain models may gain some accuracy as a result of the optimization process" [19]. In the network we have built the overall accuracy i.e., on both training and validation set remains same as that of main model but the average loss (categorical cross entropy) of optimized model decreases slightly. The main model has an average loss of 0.23845186 and the average loss of optimized model reduced to 0.23622063 (fig-7).

## 4. DEPLOYMENT

The normal microcontroller like atmega328 which is present in Arduino UNO cannot use TensorFlow library since it could not fulfil the dependencies required and due to limited oscillator frequency, RISC architecture. Hence the model is deployed on advanced microcontrollers like ARM cortex M3 and the board must be supported by TensorFlow.

We ran the quantized model in the emulated environment to analyze its performance. We used Jupyter notebook configured with python 3.6 background. Serial communication between Jupyter notebook and the Arduino UNO is established using the Pyserial library in python.

The data from the MPU6050 is received using Arduino UNO and it is transmitted serially to python environment where the model is running. After receiving the specified number of samples new line character is sent which indicates the end of samples required for classification. This data is fed to quantized neural network which performs classification in emulated environment and displays the output.

## 5. CONCLUSION AND DISCUSSION

The optimized model deployed was able to do the classification of gesture with the desired accuracy and speed there by reducing the latency and size.

The problem of size could be overcome by optimizing the neural network still the highly complicated networks with more size could not be optimized to deploy on edge devices like microcontroller and during the implementation of the network only subset of operations should be used as every operation cannot be optimized.

## REFERENCES

[1] Alexandre de Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, Yoshua Bengio. Artificial Neural Networks Applied to Taxi Destination Prediction. arXiv:1508.00021v2 [cs.LG]

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

[3] Bengio, Yoshua and Glorot, Xavier. Understanding thedifficulty of training deep feedforward neural networks.InProceedings of AISTATS 2010, volume 9, pp. 249–256, May 2010.

[4] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov Improving neural networks by preventing co-adaptation of feature detectors arXiv:1207.0580v1 [cs.NE]

[5] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980v9 [cs.LG]
[6] Geoffrey Hinton. 2012. Neural Networks for MachineLearning - Lecture 6a - Overview of mini-batch gradi-ent descent.
[7] Yang You, Igor Gitman, Boris Ginsburg. Large Batch Training of Convolutional Networks. arXiv:1708.03888v3 [cs.CV]
[8] Leslie N. Smith, Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. arXiv:1708.07120v3 [cs.LG]
[9] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167v3 [cs.LG]
[10] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S.,Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G.,Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467 [cs.DC]
[11] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.
[12] Plotly Technologies Inc. (2015). Collaborative data science. Montreal, QC: Plotly Technolo-gies Inc.
[13] Tockn. MPU6050_tockn: Arduino library for easy communicating with the MPU6050. GitHub repository. Retrieved from https://github.com/tockn/MPU6050_tockn+
[14] Arduino. (2019). ArduinoTensorFlowLiteTutorials. GitHub repository. Retrieved from https://github.com/arduino/ArduinoTensorFlowLiteTutorials
[15] TensorFlow Lite inference. Retrieved from https://www.tensorflow.org/lite/guide/inference
[16] TensorFlow model optimization. Retrieved from https://www.tensorflow.org/model_optimization/guide
[17] TensorFlow Model Optimization Toolkit — float16 quantization halves model size — The TensorFlow Blog. Retrieved from https://blog.tensorflow.org/2019/08/tensorflow-model-optimization-toolkit_5.html
[18] TensorFlow Lite for Microcontrollers. Retrieved from https://www.tensorflow.org/lite/microcontrollers
[19] Model optimization | TensorFlow Lite. Retrieved from https://www.tensorflow.org/lite/performance/model_optimization