ECE5545/CS5775 A2

Naman Makkar

TOTAL POINTS

16 / 16

QUESTION 1

1 Submit your report 16 / 16

✓ **- 0 pts** *Correct*

- 0.5 pts Clarity of plots and figures needs (some) improvement

- 0.5 pts Please include all plots in the report

- 1 pts Explanations lack depth for more than one question

- 3 pts Failed quantization tests

- 5 pts Missing pruning section

- 3 pts Pruning incomplete.

- 1 pts Pruning plots don't look correct

- 2 pts 1 day late

- 2 pts Quantization plots missing

- 2 pts 2 days late

- 5 pts Missing quantization section

- 2 pts Missing training section

- 0.5 pts 6.3 accuracy-runtime for CPU and/or MCU missing / looks incorrect

ECE5545 - Machine Learning Hardware and Systems - Assignment 2

Naman Makkar (nbm49)

March 2023

1 Part 1 - Preprocessing: Audio Recording and Visualization

1.1 Plots for the time domain, frequency domain, mel spectrogram and MFCC Spectrogram for Audio

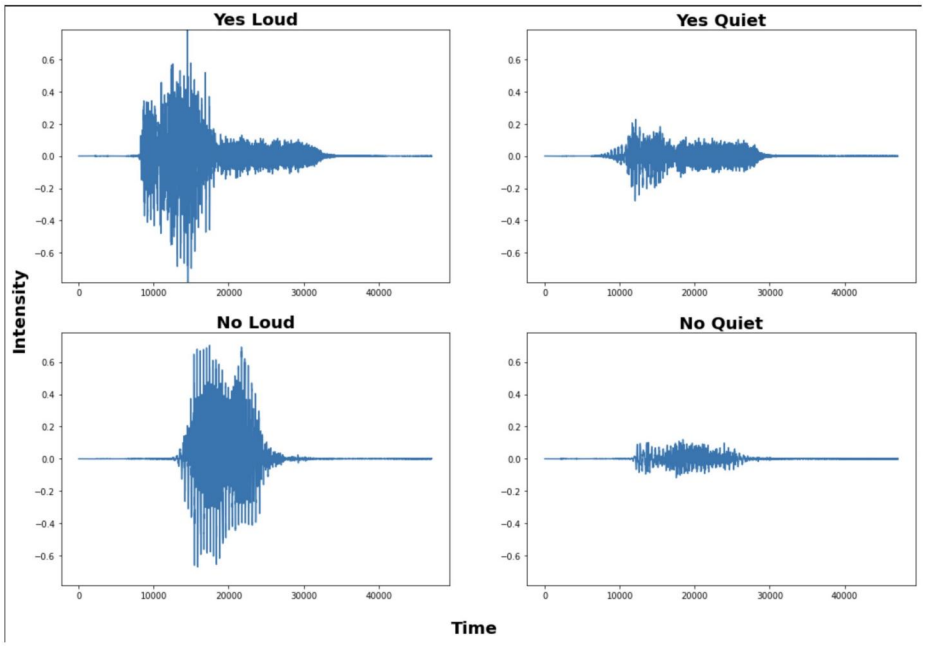

Figure 1: Plot for Time Domain

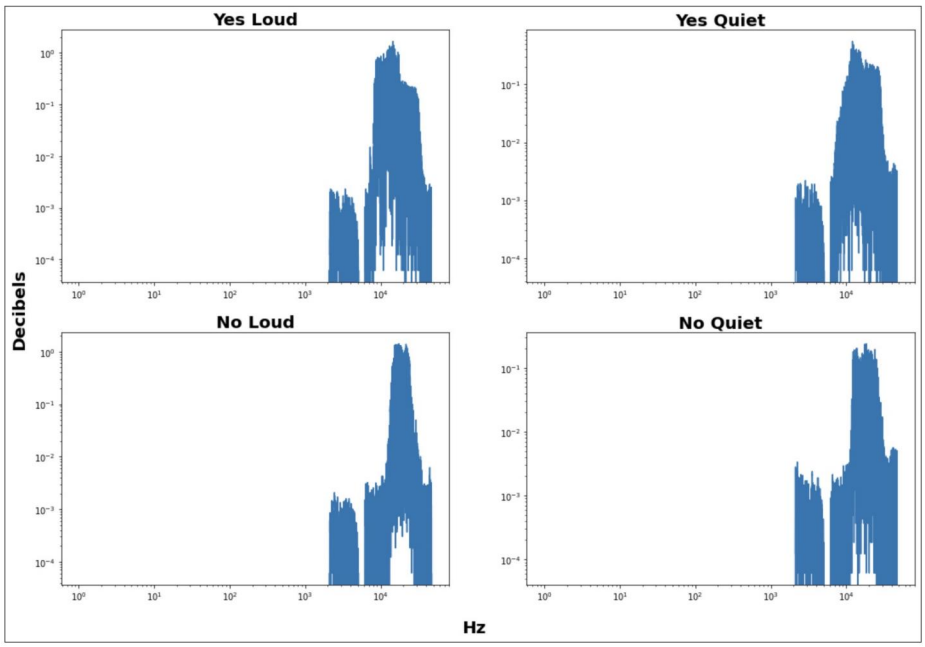

Figure 2: Plot for Frequency Domain

1.2 Preprocessing Input Audio

We preprocess audio data for the following reasons -

1. It is important to Normalize the audio data since the audio samples input to a neural network can have varying amplitudes. Normalization in preprocessing can be utilised for standardizing the amplitudes.

2. Preprocessing audio data helps us extract meaningful features from the audio waveform, these include Mel Spectrograms and Mel Frequency Cepstral Coefficients (MFCCs).

1.3 Difference between spectrograms

Differences between Mel Spectrograms and MFCCs vs Spectrograms -

1. Mel Spectrograms and Spectrograms are both representations of the frequency content of an audio signal whereas MFCCs represent the power spectrum of an audio signal. MFCCs and Mel Spectrograms differ from Spectrograms in the scale of the frequency - spectrograms make use of a linear frequency scale whereas, Mel Spectrograms and MFCCs make use of a logarithmic Mel Scale.

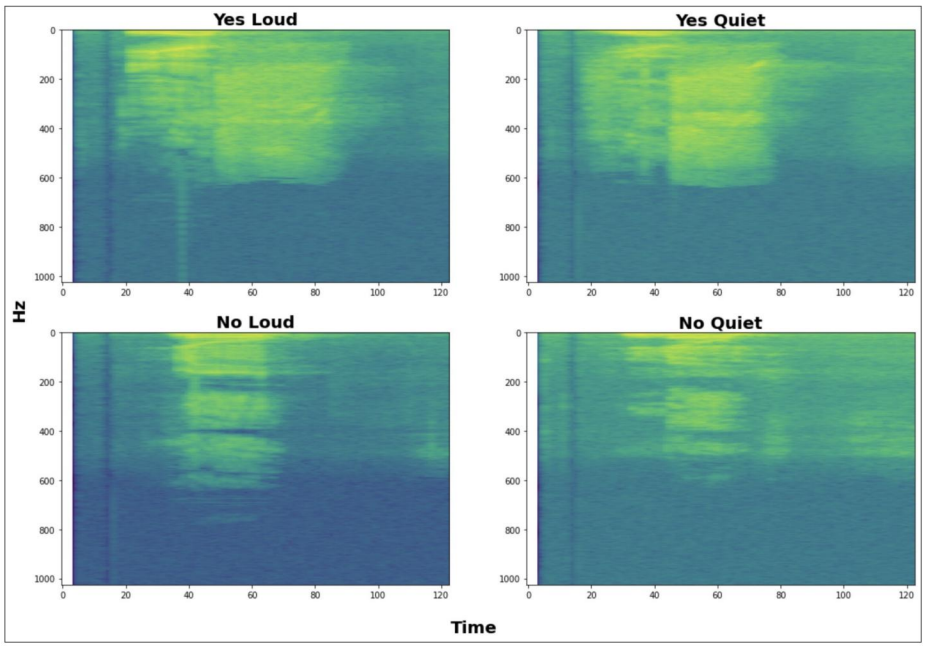

Figure 3: Plot for Spectrograms

2. MFCCs and Mel Spectrograms are utilised in tasks that require a high level of abstraction like speech recognition whereas Spectrograms are used for tasks like audio processing.

3. MFCCs and Mel Spectrograms are both less computationally expensive than Spectrograms.

Differences between MFCCs and Mel Spectrograms -

1. MFCCs are a set of coefficients which represent the power spectrum of an audio signal whereas Mel Spectrograms represent the frequency content of an audio signal.

2. MFCCs are computationally less expensive than Mel Spectrograms due to the fact that Mel Spectrograms comprise of a 2D matrix containing the magnitude of the frequency content of the audio signal, whereas the MFCC is represented as a vector of coefficients.

2 Part 2 - Model Size Estimation

2.1 Estimated Flash Usage of DNN

The estimated memory usage of the TinyConv model was calculated as 0.066608 MB = 66.608 KB. This was calculated by calculating the storage space of the

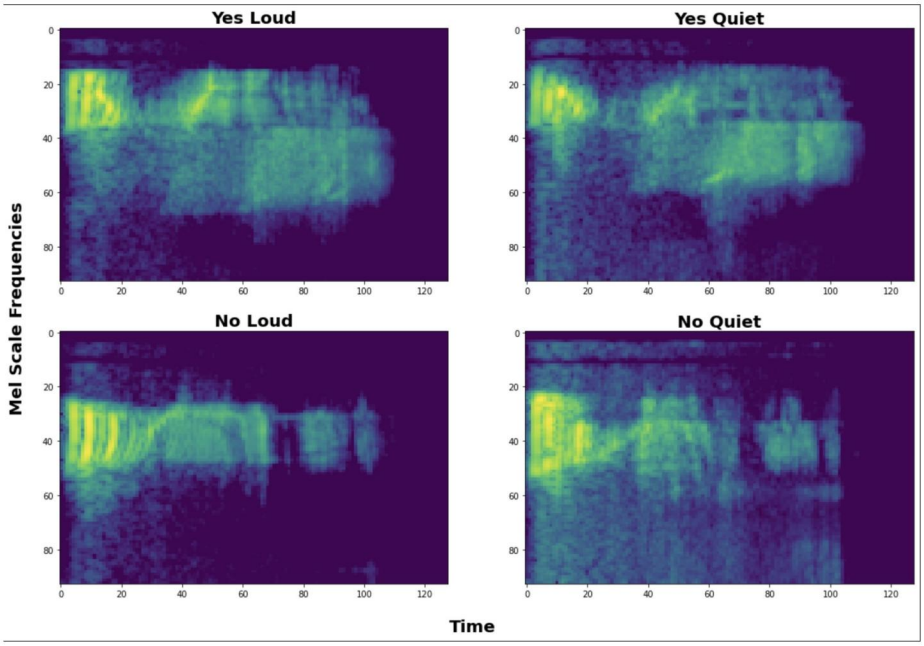

Figure 4: Plot for Mel Spectrograms

parameters of the model. This model would occupy 6.66% of the flash memory of the MCU which has a flash memory size of 1MB.

2.2 RAM usage of DNN with Batch Size = 1

The forward RAM of the TinyConv model was computed as 0.007856 MB. This was computed by calculating the storage space of the input tensor and the output tensor of the model. This accounts for 0.78% of the flash memory of the MCU.

2.3 FLOPS of the DNN

The TinyConv model had total number of floating operations as 676004, with the Conv2D layer accounting for 644000 floating operations and the Linear layer accounting for 32004 floating operations. This accounts for 0.676M FLOPS, significantly lower than the 30M FLOPS of Convolutional Recurrent Neural Network given in *Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting* `https://arxiv.org/ftp/arxiv/papers/1703/1703.05390.pdf` for the Convolutional Recurrent Neural Network (CRNN).

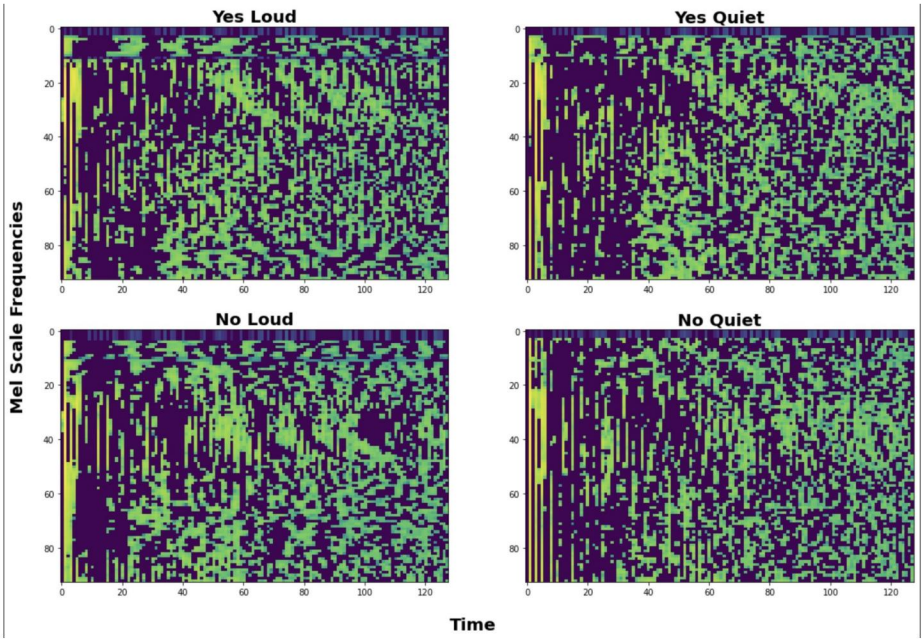

Figure 5: Plot for Mel Frequency Cepstral Coefficients

2.4 Inference time of the model on Colab CPU and GPU

The inference time of the model on Colab CPU was calculated to be **120.824ms** whereas the inference time on the Colab GPU was calculated to be **60μs**.

3 Part 3 - Training and Analysis

3.1 Accuracy of the Model on the Training, Validation and Test Sets

The TinyConv model achieved an accuracy of **90.5%** on the Test set, **90.62%** on the Validation Set and **90.9%** on the Training Set.

3.2 Plots for Training and Validation Accuracy

Figure 6 shows the plots for the Training and Validation Accuracy of the TinyConv model on the dataset.

3.3 Dataset

There are only 4 classes of keywords that are supported, these are - yes, no , #unknown and silence. The dataset consists of 10556 training samples, 1333

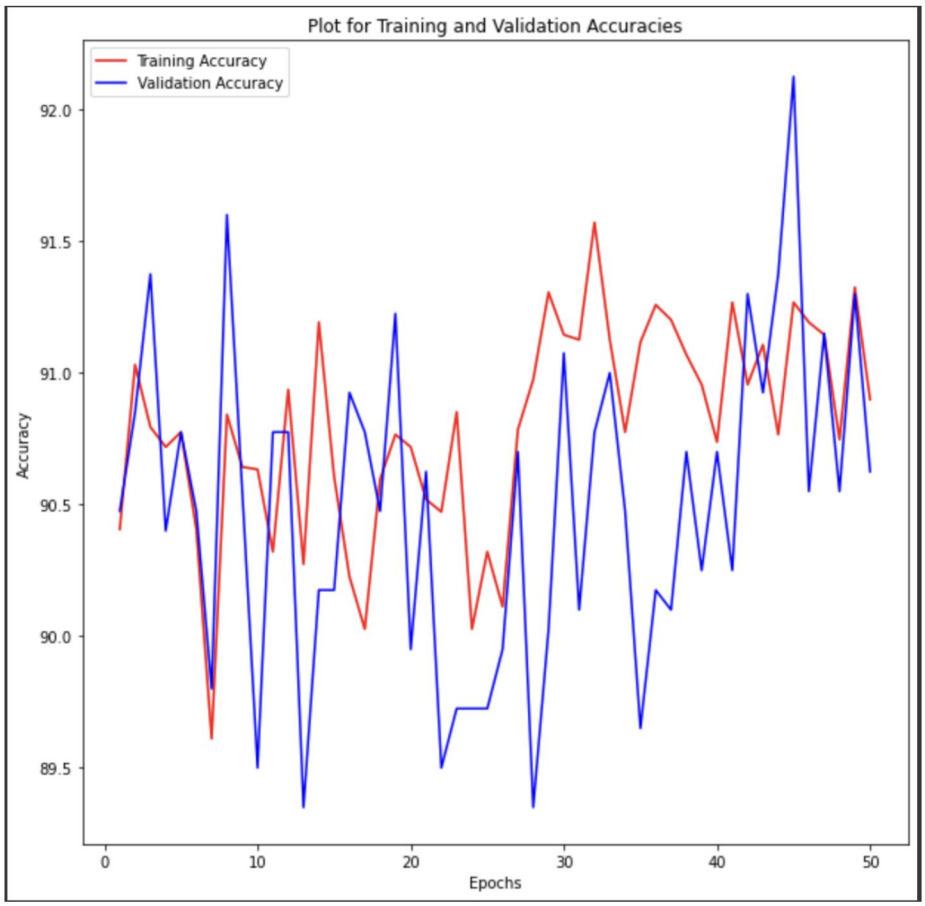

Figure 6: Plot for Training and Validation Accuracy of the TinyConv Model

validation samples and 1368 test samples.

4 Part 4 - Model Conversion and Deployment

4.1 Runtime

The Profiler outputs an average runtime of 109ms, minimum runtime of 107ms and maximum of 113ms. This can be broken down into preprocessing, model runtime and postprocessing and the profiler outputs **21ms**, **88ms** and **0ms**, respectively. Compared to the CPU which achieves a model runtime of **120.824ms** while the GPU achieved a model runtime of **60μs**

4.2 Recorded test set and accuracy

The ground truth samples tested on the MCU - **'yes', 'no', 's', 'yes', 'no', 'p', 'n','no', 'yes', 'no'** The predictions made by the model on the MCU - **'yes', 'no', 'yes', 'yes', 'no-detection', #unknown, #unknown, no-detection, 'yes', no-detection**

It was observed that the MCU was unable to detect the word 'no' 3 out of 4 times and misclassified 's' as 'yes'. It can be observed that the model deloyed on the MCU performs with a 60% accuracy on the real time test set.

The discrepancy between the accuracy achieved on the in-field test set when compared to the accuracy achieved on the training and validation set can be explained by the noisy MCU sensor which is unable to detect words like 'no' while misclassifying 's' as yes.

It could also be the case that the spectrograms generated for 's' are very similar to those generated by 'yes' leading to the model misclassifying it.

4.3 Extra Credit - Training with an additional keyword

An additional keyword, namely - **dog** was added to the wanted words in **constants.py** file and the model was trained on the dataset. It was observed that the model achieved an accuracy of 88.12% on the test set, an accuracy of 89.4% on the validation set and an accuracy of 87.97% on the training set. The plot for training and validation accuracies for extra-credit can be seen in **Figure 7**.

5 Part 5 - Quantization Aware Training

Quantization bits used for training - **2, 4, 6, 8**

5.1 Accuracy vs Quantization Bit-width for Post-Training Quantization

The plot for post-training quantization can be seen in Figure 8

5.2 Accuracy vs Bit-width for Quantization Aware Training

The plot for quantization aware training can be seen in Figure 9

5.3 Quantization Aware Training vs Post Training Quantization

It can be observed that Quantization Aware Training yields marginally better results compared to post-training quantization for 8 quantization bits, whereas post-training quantization achieves better results for 2,4,6 bits. With QAT achieving a test accuracy of **20.3%, 89.3%, 94.4%, 94.4%** for 2,4,6,8 bits

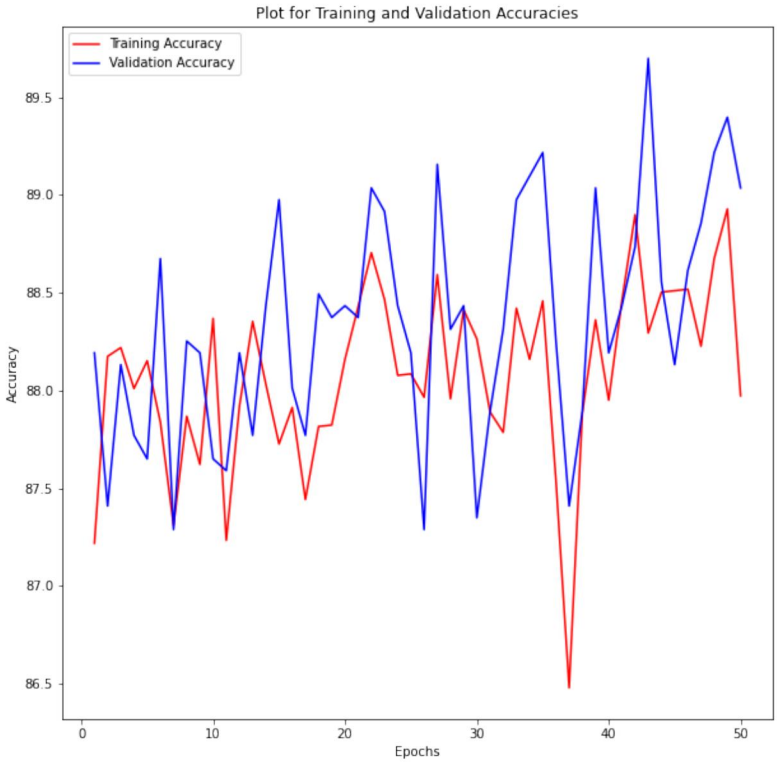

Figure 7: **Extra Credit** - Plot for Training and Validation Accuracies vs Epochs for additional keyword dog

respectively and post-training quantization achieving a test accuracy **20.3%, 90.1%, 95.1%, 93.8%** for 2,4,6,8 quantization bits respectively.

6 Part 6 - Pruning

6.1 Unstructured Pruning

Figures 10 and 11 show the plots for Unstructured Pruning. Pruning was carried out with the help of **torch.nn.utils.prune.l1_unstructured**

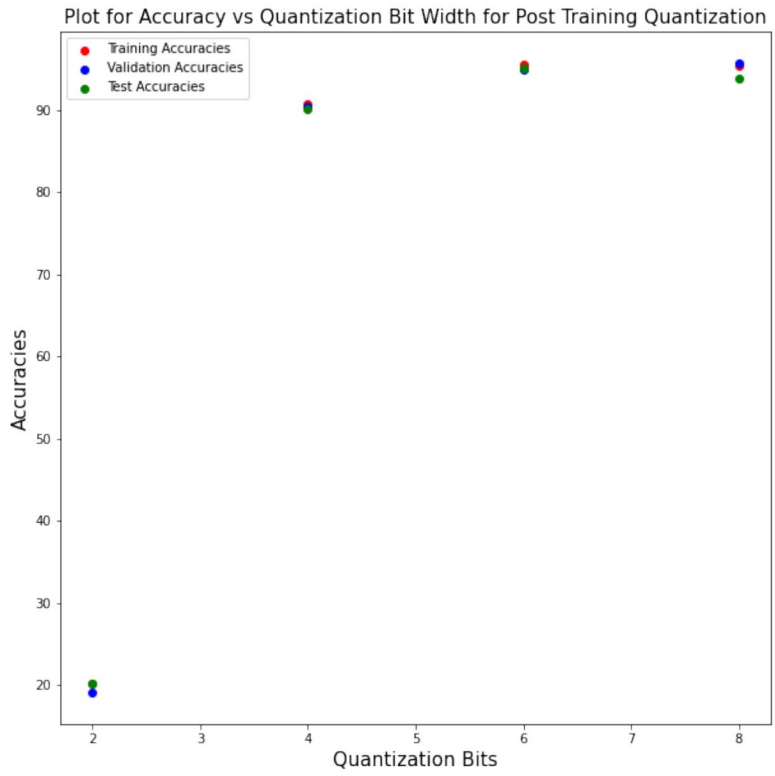

Figure 8: Plot for Accuracy vs Bit Width for Post Training Quantization

6.1.1 Speeding up computation with unstructured pruning

Unstructured pruning reduces the number of parameters of a model. It removes a certain number of weights and connections of the model architecture with the help of a threshold, this leads to a reduction in the FLOPs of the model architecture and a reduction in the number of computations needed to be carried out for both training and inference. Additionally, this reduces the storage space required for a model architecture.

6.1.2 Difference between L1, L2 and L-infinity norms and applications in Pruning

The L1 norm of a vector is the sum of the absolute values of its elements, whereas the L2 norm of a vector is the square root of the sum of the squared

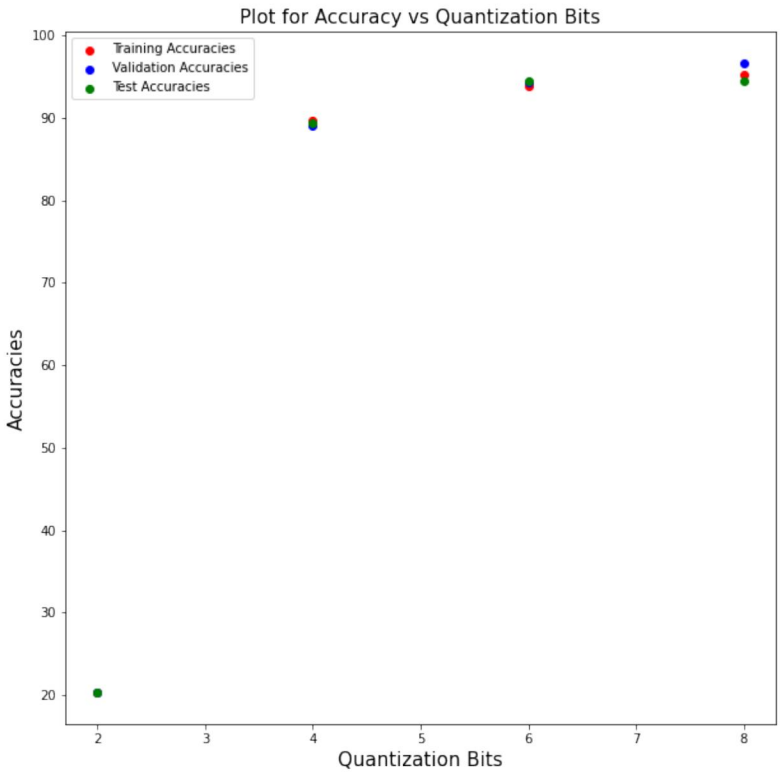

Figure 9: Plot for Accuracy vs Bit Width for Quantization Aware Training (QAT)

values of its elements, while the L-infinity norm is the maximum absolute value of the vector.

L1 norm is the most common for pruning since it encourages sparsity due to the fact that it penalizes the weights according to their magnitude. This is preferable to L2-norm since the L2 norm penalizes each weight equally, regardless of their magnitude. Similarly, the L-infinity norm does not encourage sparsity either since it cannot distinguish between magnitudes of weights, since it simply returns the maximum absolute value of a vector, it penalizes all weights equally regardless of the magnitude.

Therefore, L1-norm is the best for pruning.

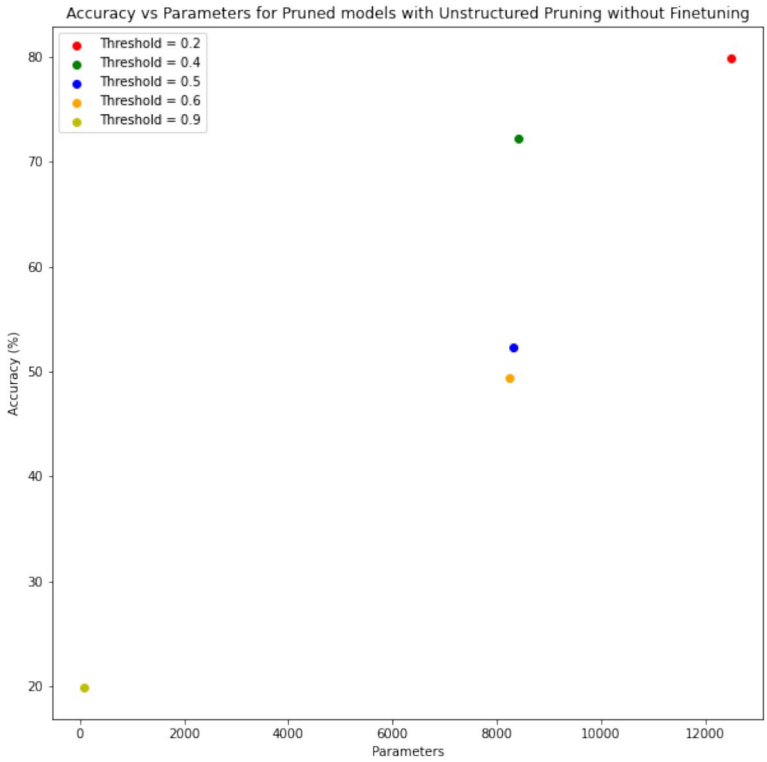

Figure 10: Parameters vs Accuracy for Unstructured Pruning models **without Finetuning**

6.2 Structured Pruning

Plots for Accuracy vs Parameters with structured pruning can be observed in **Figures 12 and 13**. Whereas plots for Accuracy vs FLOPs can be observed in **Figures 14 and 15**. While plots for Accuracy vs CPU Runtime can be observed in **Figures 16 and 17**, and those for Accuracy vs MCU Runtime can be observed in **Figures 18 and 19**.

6.2.1 Accuracy vs Parameters

6.2.2 Accuracy vs FLOPs

6.2.3 Accuracy vs Runtime on Dektop CPU

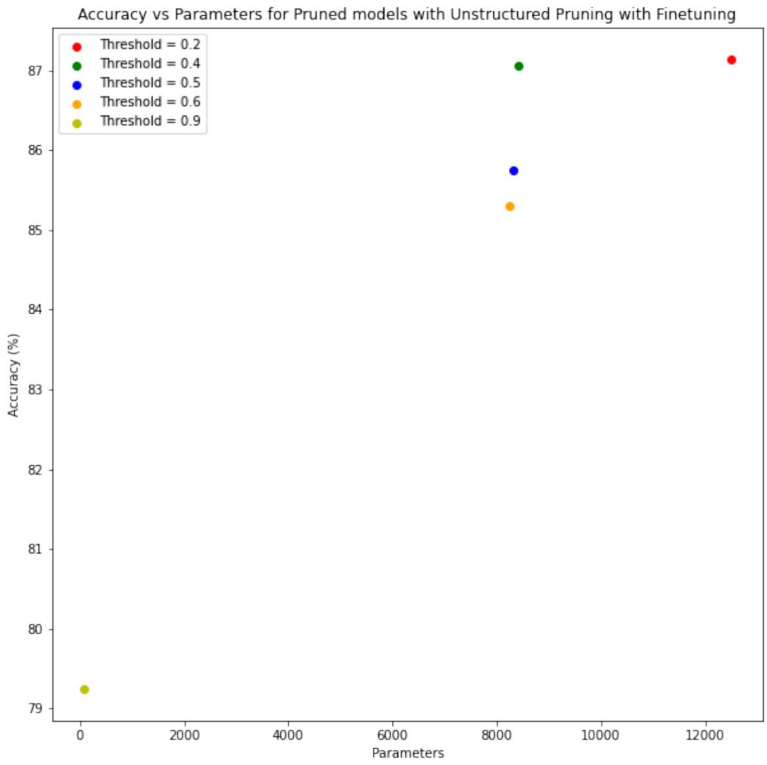

Figure 11: Parameters vs Accuracy for Unstructured Pruning models with Fine-tuning

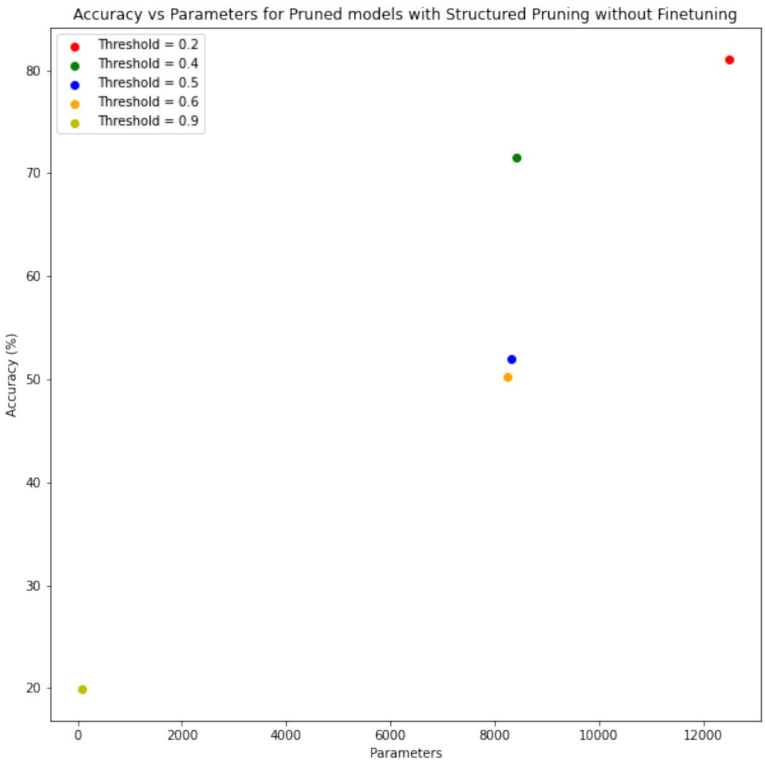

Figure 12: Parameters vs Accuracy for Structured Pruning models **without Finetuning**

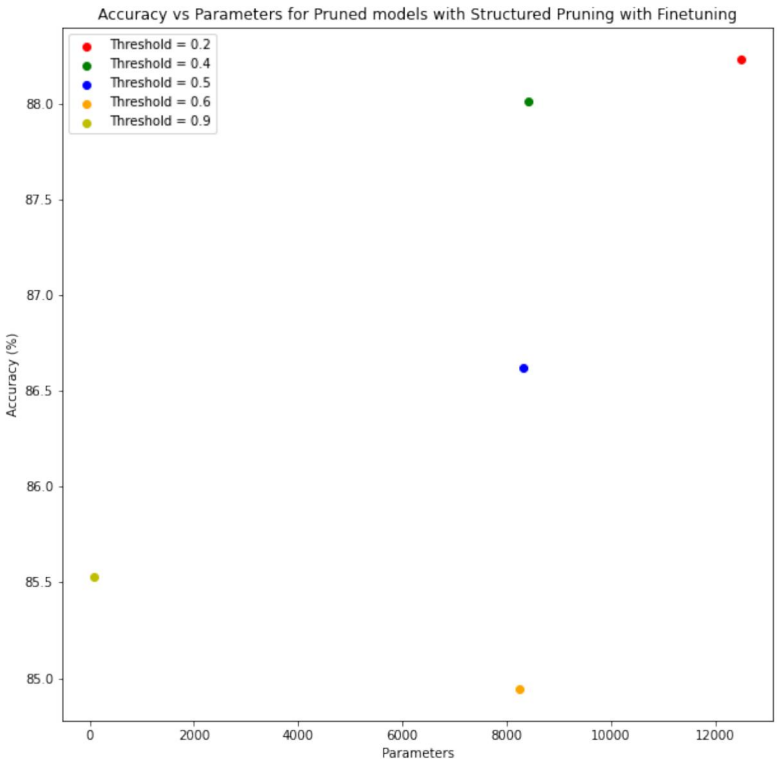

Figure 13: Parameters vs Accuracy for Structured Pruning models with Fine-tuning

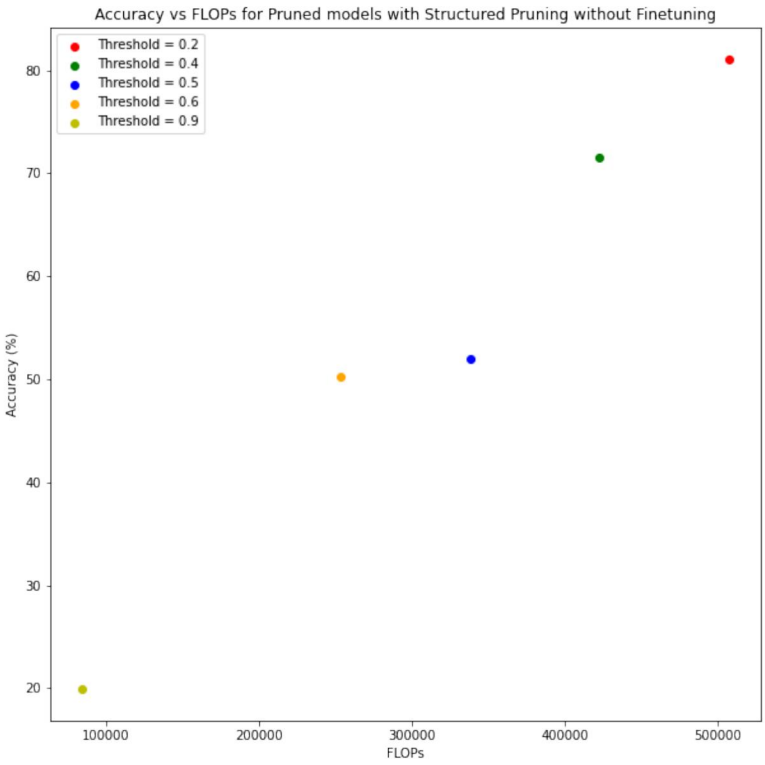

Figure 14: FLOPs vs Accuracy for Structured Pruning models **without Finetuning**

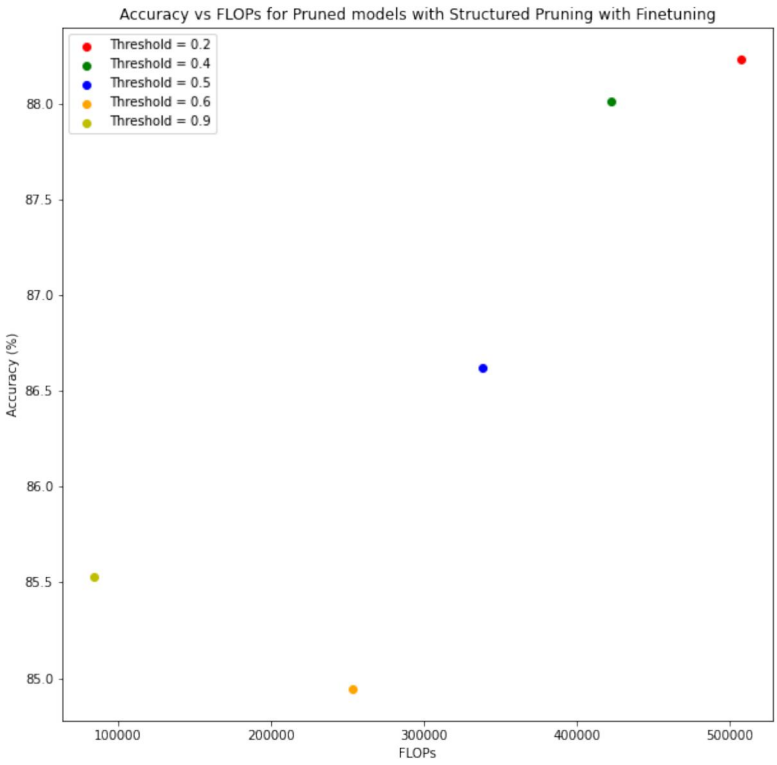

Figure 15: Parameters vs Accuracy for Structured Pruning models with Fine-tuning

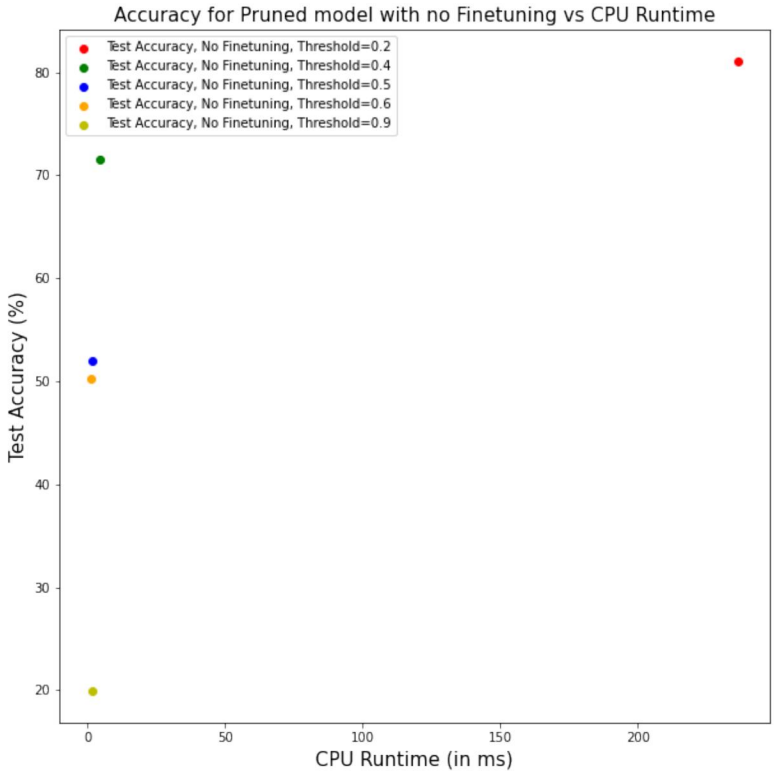

Figure 16: Accuracy vs CPU Runtime in ms for the pruned models without finetuning can be observed

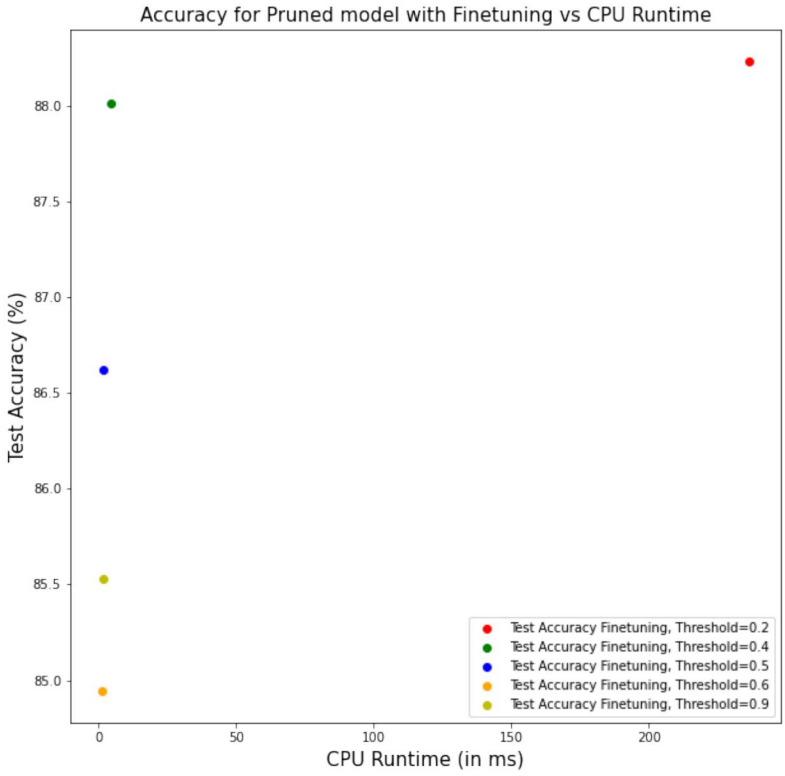

Figure 17: Accuracy vs CPU Runtime in ms for the pruned models with fine-tuning

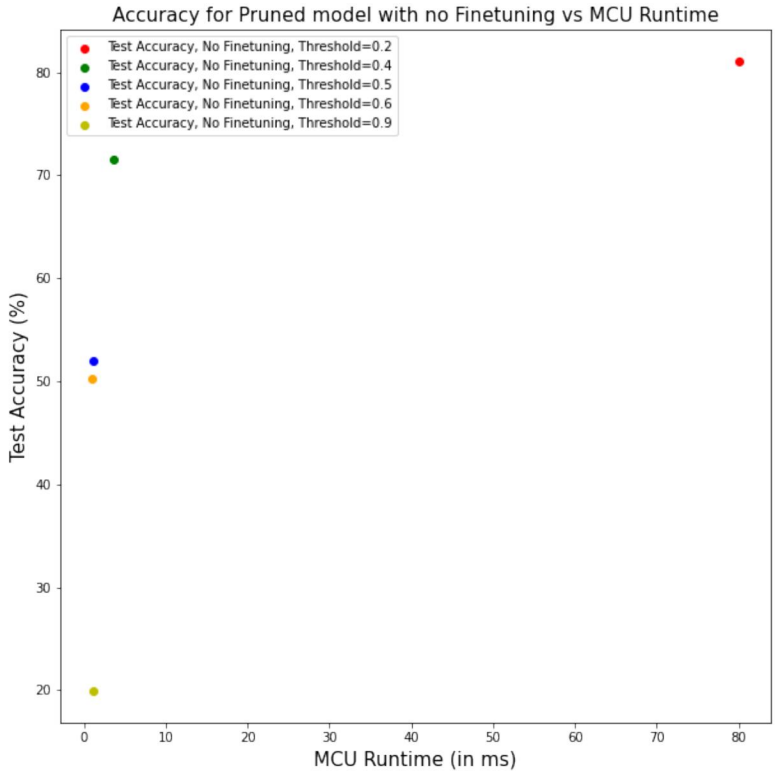

Figure 18: Accuracy vs MCU Runtime in ms for the pruned models with no finetuning

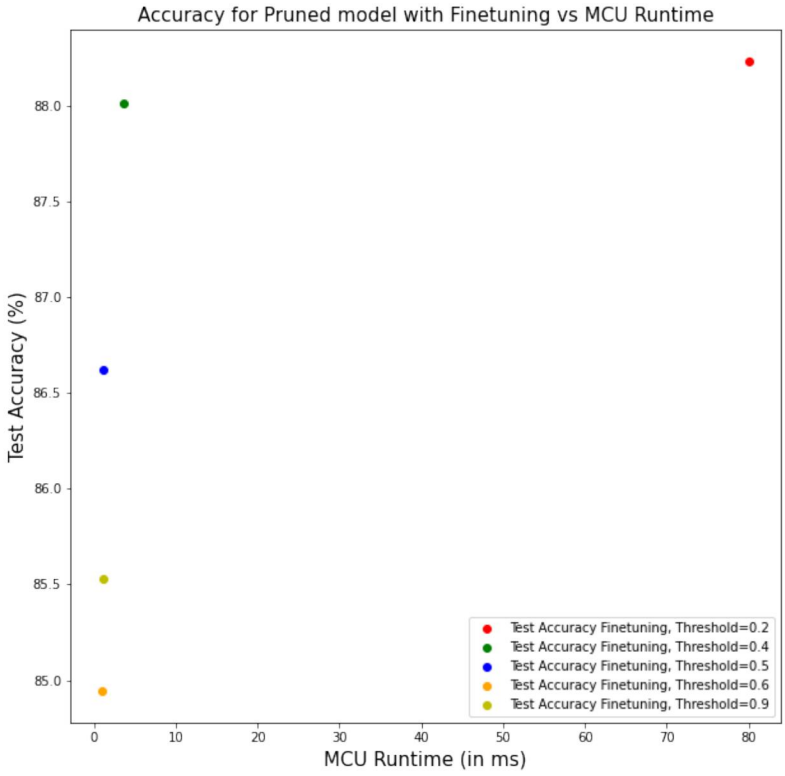

Figure 19: Accuracy vs MCU Runtime in ms for the pruned models with fine-tuning

1 Submit your report 16 / 16

✓ **- 0 pts** *Correct*

- 0.5 pts Clarity of plots and figures needs (some) improvement

- 0.5 pts Please include all plots in the report

- 1 pts Explanations lack depth for more than one question

- 3 pts Failed quantization tests

- 5 pts Missing pruning section

- 3 pts Pruning incomplete.

- 1 pts Pruning plots don't look correct

- 2 pts 1 day late

- 2 pts Quantization plots missing

- 2 pts 2 days late

- 5 pts Missing quantization section

- 2 pts Missing training section

- 0.5 pts 6.3 accuracy-runtime for CPU and/or MCU missing / looks incorrect