COMS 6998 Final Project Report

Model Compression and Acceleration Analysis of different Models

Submitted by: Adit Deshmukh (avd2133), Vani Jain (vj2245)

1. Introduction

Deploying Deep Learning models on edge devices faces multiple challenges:

a. Deep Learning models are computationally expensive and memory intensive
b. Edge devices do not have the hardware resources to run inference on models quickly
c. Most use cases for edge device deployment require real time inference in order to be useful e.g. self driving cars

This necessitates using techniques that make the models less resource hungry while also increasing their speed. Table 1 summarises the different techniques that can be used and the types of applications they are applicable to.

Category Name	Description	Applications	More details
Parameter pruning and quantization	Reducing redundant parameters which are not sensitive to the performance	Convolutional layer and fully connected layer	Robust to various settings, can achieve good performance, can support both train from scratch and pre-trained model
Low-rank factorization	Using matrix/tensor decomposition to estimate the informative parameters	Convolutional layer and fully connected layer	Standardized pipeline, easily to be implemented, can support both train from scratch and pre-trained model
Transferred/compact convolutional filters	Designing special structural convolutional filters to save parameters	Convolutional layer only	Algorithms are dependent on applications, usually achieve good performance, only support train from scratch
Knowledge distillation	Training a compact neural network with distilled knowledge of a large model	Convolutional layer and fully connected layer	Model performances are sensitive to applications and network structure only support train from scratch

Table 1. Optimization techniques

2. Techniques Used

a. Quantization

Quantization compresses the original network by reducing the number of bits required to represent each weight. It is a helpful post processing technique for deep learning models that makes them more suitable for deployment on edge devices. Table 2 lists the different quantizations supported by tensorflow and their benefits.

Technique	**Benefits**	**Hardware**
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

Table 2. Quantization techniques

b. Pruning

Pruning refers to the process of “trimming” insignificant weights from the model. Magnitude based pruning zeroes out model weights during training to achieve sparsity. Weights that do not affect the output significantly are pruned. Pruning happens iteratively over epochs. Pruning can significantly reduce model size without losing performance when the model has excess “capacity”. That is when a very powerful model is used on an easy task, it can be made sparse to a large extent. This helps to reduce the “effective” model size. While the model size does

not change, due to the sparsity introduced, the size of the model in compressed form reduces. This makes it cheaper to transfer the model.

c. **Weight Clustering**
Weight clustering groups weights together and reduces the unique weight values in a model. It groups together the weights in any layer into N clusters and then uses the centroid of each cluster to represent the weights belonging to that cluster. As in pruning, the benefit is seen not in the actual model size but in the compressed model size. As of now, tensorflow does not support latency improvements during inference for pruned and clustered models. However, it is likely in the future.

3. Model Architecture

a. **MobileNet**
MobileNet is built on depth-wise separable convolutions except the first layer, which is a full convolutional layer. All but the last layers are followed by ReLU non-linearity and batch normalization. The final layer is a fully connected layer without any non-linearity and feeds to the softmax for classification. For downsampling, strided convolution is used for both depthwise convolution as well as for the first fully convolutional layer. The total number of layers for mobilenet is 28 considering depthwise and pointwise convolution as separate layers.

b. **EfficientNet**
EfficientNet is a CNN architecture and scaling method. It uses a compound coefficient to scale depth, width and resolution. Instead of arbitrarily scaling the coefficients, EfficientNet uniformly scales network width, depth and resolution. EfficientNet is trained on more than 1 million images from the ImageNet dataset and can classify images into 1000 object categories.

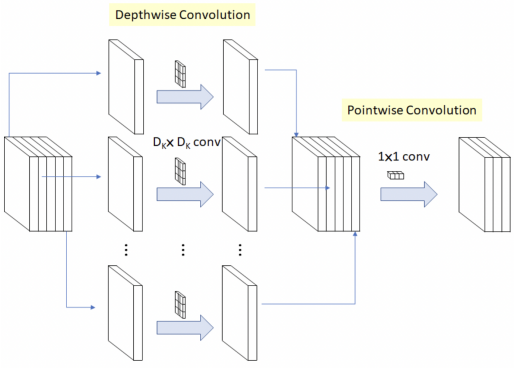

Fig 1. MobileNet Architecture

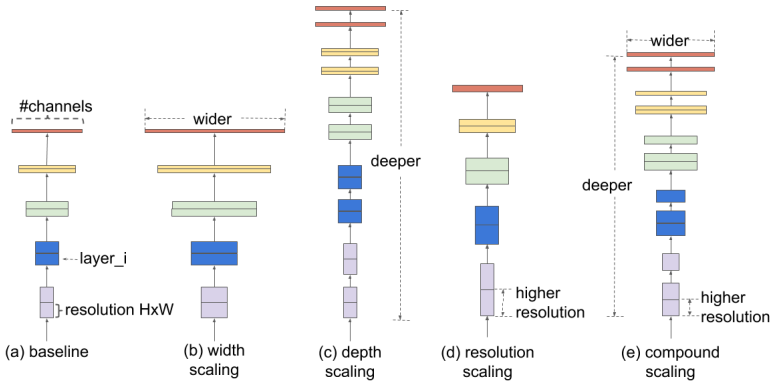

Fig 2. EfficientNet Architecture

4. Experiments Performed

The aim of this project is to survey the different model compression and acceleration techniques and see how they affect the accuracy, model size and inference time.
We also aim to see whether this behavior is different for different network architectures. Here we consider two common architectures used as backbones in many vision tasks and also suitable for deployment on mobile devices. We consider MobileNet and EfficientNet.

The first task we consider is the image classification task. Usually models are pre-trained for classification and then used for downstream application like object detection. The performance of these backbone networks and the quality of features they can produce affects the downstream performance. We evaluate the impact of the different compression and acceleration techniques on the selected models for classification on CIFAR100. We choose CIFAR100 as it has 100 classes which is similar to the 90 classes in COCO. Our aim is to analyze the data and come up with the best compression technique for SSD pretrained on COCO under the assumption that we do not have access to a representative dataset to be able to conduct all the experiments on object detection. We carry out a generalizable analysis which can be extended to other downstream tasks. We also deploy several models on Android to get the "real world" estimate of inference time.

5. Experiment Results

a. Results after Quantization

	Without	Dynamic	Full Integer	Float16
MobileNet	58.74%	58.56%	57.14%	58.61%
EfficientNet	56.55%	55.74%	51.29%	56.58%

Table 3. Accuracy Table after Quantization

	Without	Dynamic	Full Integer	Float16
MobileNet	1.2MB	383KB	415KB	619KB
EfficientNet	17MB	4.4M	5.3M	8.4M

Table 4. Model Size Table after Quantization

b. Accuracy

Table 5 shows how accuracy changes with changing sparsity while pruning. For MobileNet, we observe a decrease in accuracy as we drop more weights with increase in sparsity. However, for EfficientNet we observe that accuracy is highest at 0.4 sparsity. This is because the EfficientNet is over-parameterized for the task.

	Base Model	**Integer Quantization**	**Pruning at 0.2 Sparsity**	**Pruning at 0.4 Sparsity**	**Pruning at 0.6 Sparsity**	**Pruning at 0.8 Sparsity**	**Pruning and Quantization**
MobileNet	61.55%	61.28%	61.34%	60.47%	60.43%	54.67%	60.27%
EfficientNet	56.55%	51.29%	50.59%	51.10%	48.17%	45.30%	49.19%

Table 5. Accuracy Table

c. Model Compression

From table 6 below we can see that with increase in sparsity, the model size decreases. This is because increase in sparsity implies that more weights are dropped, hence reducing the model size.

	Base Model	**Integer Quantization**	**Pruning at 0.2 Sparsity**	**Pruning at 0.4 Sparsity**	**Pruning at 0.6 Sparsity**	**Pruning at 0.8 Sparsity**	**Pruning and Quantization**
MobileNet	1.17MB	3.8MB	0.98MB	0.81MB	0.61MB	0.40MB	0.31MB
EfficientNet	18MB	5.3MB	14.38MB	8.99MB	8.90MB	5.75MB	4.33MB

Table 6. Model Compression Table

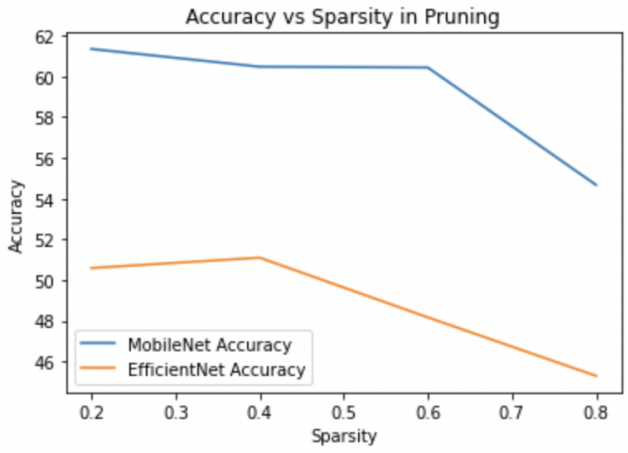

Fig 3. Accuracy vs Sparsity Graph

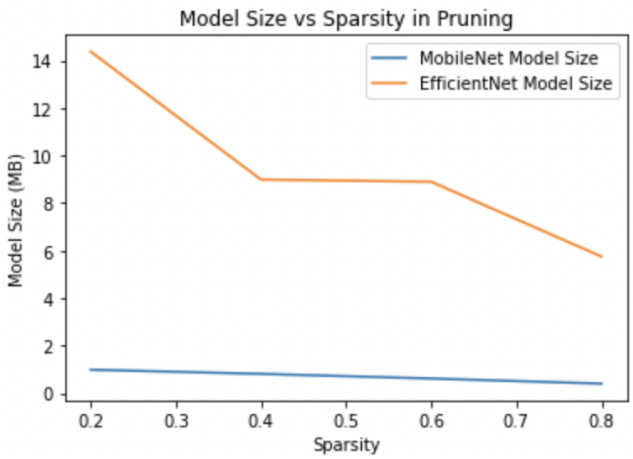

Fig 4. Model Size vs Sparsity Graph

6. Demo

a. MobileNet

We deployed MobileNet base model and quantized model on edge device to obtain the inference times at different threads. From table 7 below, we observe the inference time reducing with increasing threads. We also observed decrease in inference time while using a quantized model.

	1 thread	2 threads	3 threads	4 threads
Base Model	156ms	112ms	76ms	67ms
Integer Quantized	104ms	71ms	55ms	47ms

Table 7. MobileNet Inference Table

From the graph we can observe that the inference time decreases with quantization. We also observe that increasing the number of threads reduces the inference time.

b. EfficientNet

We deployed EfficientNet base model and quantized model on edge device to obtain the inference times at different threads. From table 8 below, we observe the inference time reducing with increasing threads. We also observed decrease in inference time while using a quantized model.

	1 thread	2 threads	3 threads	4 threads
Base Model	172ms	127ms	85ms	73ms
Integer Quantized	113ms	84ms	62ms	51ms

Table 8. EfficientNet Inference Table

From the graph we can observe that the inference time decreases with quantization. We also observe that increasing the number of threads reduces the inference time.

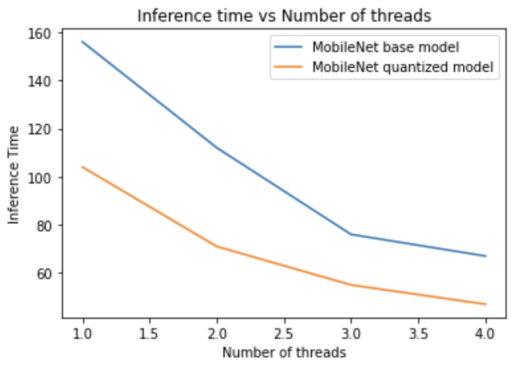

Fig 5. MobileNet Inference Time Graph

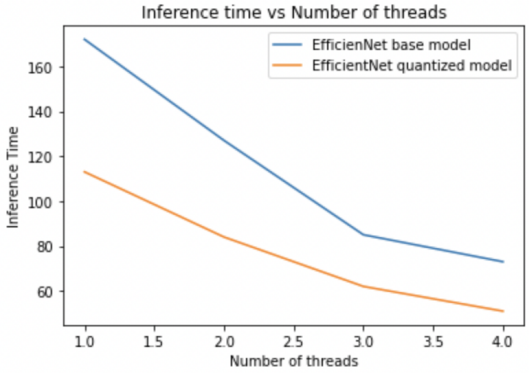

Fig 6. EfficientNet Inference Time Graph

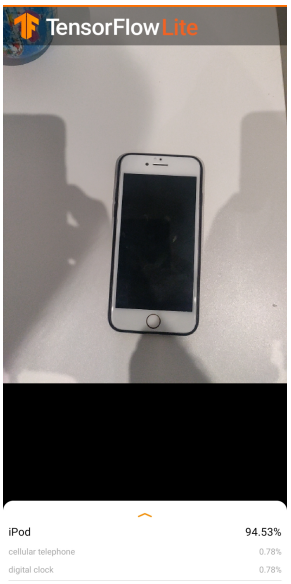

Fig 7. MobileNet Base

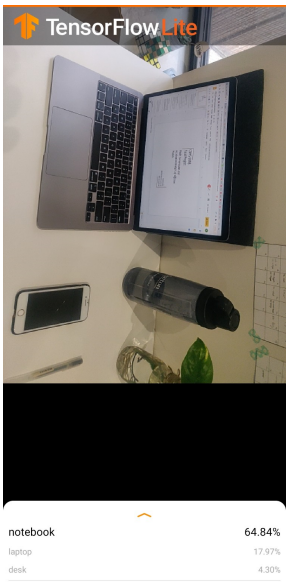

Fig 8. MobileNet Quantized

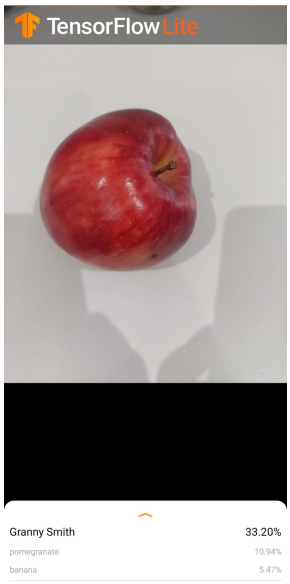

Fig 9. EfficientNet Base

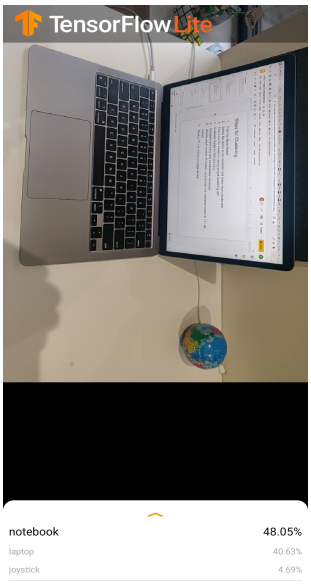

Fig 10. EfficientNet Quantized

c. Object Detection

	Base Model	Integer Quantization	Pruning(0.2)
MobileNet	28.2	27.8	26.3

Table 9. Object Detection Accuracy Table

	Base Model	Integer Quantization	Pruning(0.2)
MobileNet	13MB	4.4MB	11.2MB

Table 10. Object Detection Accuracy Table

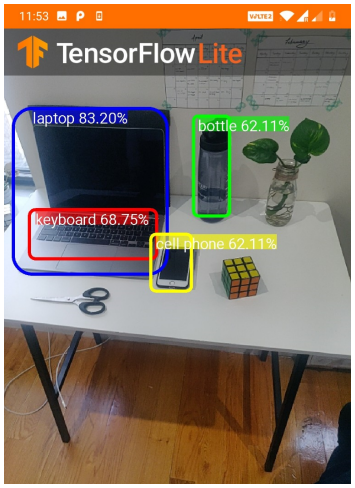

Fig 11. MobileNet Base Model

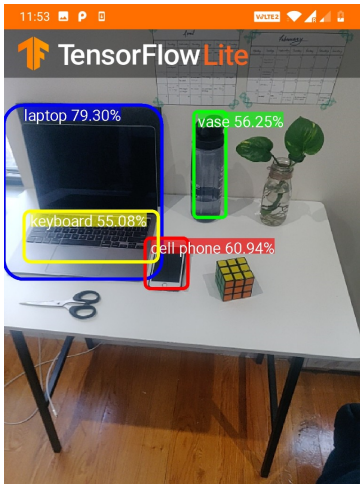

Fig 12. MobileNet Quantized Model

7. Observations

a. We did not observe a drastic drop in accuracy for the classification model even after quantization and significant level of pruning.
b. While we did not observe a 2x-3x speedup for quantized model, this was down to the fact that models were being tested using Mobile CPU and not GPU. We still observed significant speedup for the quantized models.
c. Clustering resulted in no learning as the number of weights is too large (in millions) to be clustered into a small number of clusters. Clustering gave good results for a small model trained on MNIST.
d. Quantization and pruning together resulted in even smaller model with very little drop in accuracy.
e. We found that even at high sparsity levels (0.6), the drop in accuracy was not very significant (less than 2% in most cases).
f. While MobileNet and EfficientNet are designed to be fast, the inference time we observed per image for float models was not very fast.
g. The inference was much faster when more threads were used.
h. The effect of quantization was similar to using an additional thread on the non quantized model.
i. The quantized models with multiple threads gave real time frame rates even on a CPU.
j. In some experiments, inducing sparsity resulted in slightly higher accuracy.

8. Inference

a. Quantization can be used to get significant reduction in model size as well as speedup during inference at very little loss of accuracy.
b. Pruning can be effectively used for over parameterized models. As we saw in our experiments, the accuracy for MobileNet went down with increased sparsity. However, for EfficientNet which has a higher number of parameters, the accuracy actually increased with increase in sparsity. This is most likely due to EfficientNet being over parameterized for the task at hand.
c. Clustering is not a practical approach for large models as all the thousands of weights of any layer cannot be clustered into a small number of clusters.

9. Conclusion

From our experiments we conclude that optimization techniques, in our case, pruning and quantization, are very efficient for reducing the size of the model that we want to deploy on the edge devices. These techniques can be fine tuned based on the requirement at hand and can help us achieve models that are less resource hungry while ensuring that accuracy is not drastically compromised. By deploying the models on the edge device, we were also able to observe the results of our experiments and compare the performance of different models on our edge device.

10. Future Work

a. We have analyzed models which are developed for edge devices. Future work would be to analyze a larger range of models commonly used as backbone networks such as ResNets.
b. While we have observed the impact on Convolutional Networks, future analysis could also concentrate on domains other than Computer Vision, which use different types of models e.g. LSTMs and Transformers.
c. There is also significant work being carried out to optimize specialized hardware for the deployment of deep learning models.