Optimisation of Neural Networks through Pruning and Quantisation

ENSC 424 Group 6:
Fahad Mohamed - 301591611
Fahad Yussuf - 301592480
Lucas Dai - 301559271

Table of Contents

1.0 Introduction

As neural networks grow in complexity and accuracy, their performance relies heavily on model parameters, which take a lot of memory and computational power. For example, the very popular GPT-3 includes up to 175 billion parameters [1], a massive increase from previous models such as ResNet and LeNet. This trend leads us to assume that with improved models, the number of parameters will also continue to increase. The high cost and memory constraints are even more apparent when these models are deployed on devices with limited computational resources, energy, and bandwidth, such as mobile phones [2] [3]. This necessitates a need to optimize these networks, making them lightweight, all while still maintaining their accuracy. In this project, we sought to improve this by applying quantization and pruning techniques to a ResNet CNN model fine-tuned on CIFAR-10, not only on their own but also in tandem as we seek to find the best trade-off between performance and accuracy.

2.0 Baseline Models

2.1 ResNet-18 Design and Implementation

To establish a strong baseline for the CIFAR-10 classification, we fine-tuned a pre-trained ***ResNet18*** model on ImageNet. We aimed to balance training speed, model size, and accuracy throughout the process before applying any compression techniques.

2.1.1 Dataset Preparation

We implemented a preprocessing and augmentation pipeline to improve generalization and ensure CIFAR-10 was compatible with the pretrained weights:

- ***Resizing*** the images to 224*224, matching the expected input by ResNet18
- ***Augmentations*** during training and ***normalisation*** using ImageNet mean and standard deviation to ensure CIFAR-10 dataset features were compatible with pretrained weights that were based on ImageNet

2.1.2 Model Design (Baseline Architecture)

We selected ResNet18 with pretrained ImageNet weights. To fine-tune the model to the CIFAR-10 dataset, we started by freezing all the layers and only left the classifier (FC) unfrozen to allow learning of the dataset features. However, this approach did not give us sufficient accuracy.

We, therefore, decided to also unfreeze layers 3 and 4 in addition to the classifier so as to be able to achieve deeper fine-tuning. We also replaced the final fully connected layer with a new linear head to match the 10 classes in CIFAR-10 and not the 1000 from ImageNet that the model was trained on, thereby reducing training cost while improving accuracy to >90%.

2.1.3 Training Configuration

The training loop was designed with a learning rate of ***1e-4,*** updating only the unfrozen layers. The loss function was implemented via cross-entropy for multi-class classification. A single epoch was used in this baseline experiment for initial metrics. We stayed with one epoch, as the accuracy was >90%, and we decided that the computational cost of more epochs was not worth the extra few % in accuracy.

2.1.4 Evaluation and Performance Metrics

We implemented:

- Validation set accuracy computation by utilizing a no-grad evaluation loop. This helped reduce computational cost and speed up the evaluation process
- A lightweight latency test on both the CPU and GPU, to determine average inference time
- A counter to count trainable and total model parameters

2.1.5 Results and Output

Along with saving the fine-tuned model to Google Drive, we also reported on the parameters in the evaluation section, as can be seen in the image below:

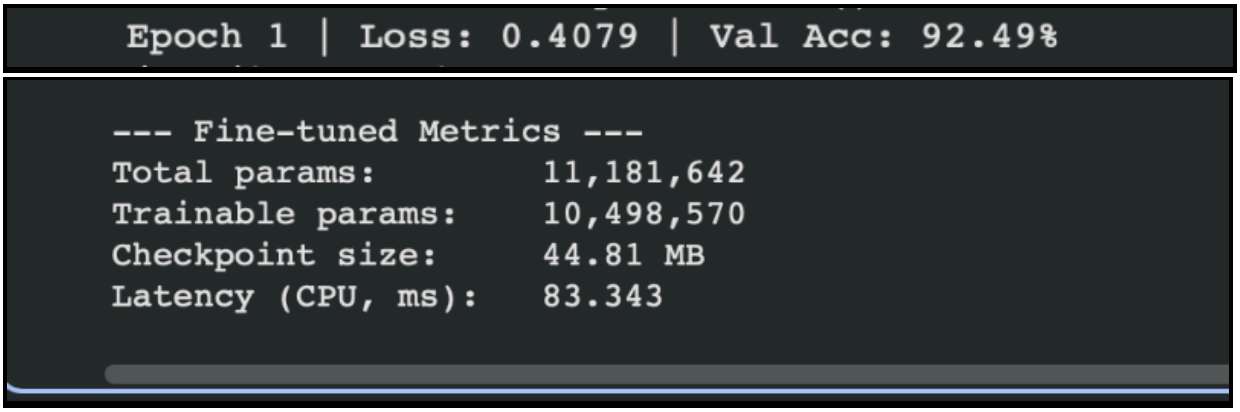

Figure 1 : ResNet18 baseline results

2.1.6 Analysis

As can be seen in Figure 1 above, **93%** of the total parameters were trainable, the model size is **44.81MB,** and the inference latency was **83.343 ms.** These values are in line with our earlier assumptions regarding computational cost and model accuracy pre-compression.

2.2 VGGNet Design and Implementation

We adapted an ImageNet-pretrained VGG16-BN model by retraining it on CIFAR-10. This gives us a good trade-off between accuracy, computational cost, and model complexity before any compression techniques are applied.

2.2.1 Model Design (Baseline Architecture)

For our baseline, we used a VGG16 network with batch normalization (VGG16-BN) initialized with ImageNet pretrained weights. We first adapted the model to the CIFAR-10 dataset by replacing the last fully connected layer with a new linear layer with 10 outputs. This helps align the classifier with the CIFAR-10 label space.

At first, we tried only changing the new classification and keeping all earlier layers; however, the model could not fully adapt to the new dataset. To make the baseline stronger, we then allowed deeper VGG blocks and the classifier to update during training. This allowed for a more accurate baseline model while still allowing it to utilize the original ImageNet pretraining.

2.2.2 Training Configuration

We trained the VGG16-BN baseline using a learning rate of 1*10^(-4) and optimized only the unfrozen layers. Training used cross-entropy loss for the CIFAR-10 classification task. For the baseline, we ran a single epoch; since it already reached over 90% accuracy, additional epochs would have increased training time and usage for only marginal gains in performance.

2.2.3 Evaluation and Performance Metrics

We implemented:

- No-gradient evaluation loop to compute validation accuracy efficiently.
- A lightweight latency benchmark on both the CPU and GPU to estimate average per-image inference time.
- Added a parameter counter to report both the total number of parameters and the subset that is trainable after freezing layers.

2.2.4 Results and Output

The results from our implementation are shown below.

```
--- BASELINE VGG16-BN (FP32) RESULTS ---
Accuracy:            90.73%
Model Size:          537.31 MB
Total Params:        134,309,962
Trainable Params:    134,309,962
Latency (GPU):       10.227 ms
Latency (CPU):       355.767 ms
```

Figure 2: VGGNet Baseline Results

2.2.5 Analysis

From Figure 2 above, our VGG16-BN baseline in FP32 achieves **90.73%** test accuracy, with a model size of **537.31MB** and **134,309,962** total parameters. The measured inference latency is **10.227 ms** on the GPU and **355.767 ms** on the CPU. These results confirm that the uncompressed baseline we inferred has strong accuracy before compression.

3.0 Compression Techniques

After establishing the baseline fine-tuned ResNet-18 model, we applied structured pruning, Post-Training-Quantization (**PTQ**), and Quantization Aware Training (**QAT**) to reduce model size and improve inference latency. These methods offer varying degrees of compression and accuracy, helping us to evaluate performance, cost, and accuracy trade-offs. We will apply the techniques discussed below to both the fine-tuned ResNet18 and VGGNet models, highlighting the effects on both.

3.1 Dynamic Quantization (PTQ)

This is the simplest form of post-training quantization, where the model's weights are transformed from floating-point to int8 representations and the activations are left in floating-point form and quantized dynamically during runtime.

This method did produce a slight reduction in model weight, on ResNet18, from **44.81MB** to **44.79MB,** while accuracy and latency were very similar to the fine-tuned model, as can be seen in the image below:

Figure 3: ResNet18 Dynamic PTQ

This is in line with expectations, as dynamic PTQ does not aggressively compress the model.

When applied to VGGNet, the baseline achieved **90.73%** test accuracy with a model size of **537.31MB** and a latency of **355.77 ms.** After dynamic PTQ, the model size dropped to **71.81MB,** but the accuracy dropped to **11.09%**, and there was an increase in latency to **362.933ms**.

Figure 4: VGGNet Dynamic PTQ

Since we replaced the classifier with a randomly initialized classifier without retraining, this results in near-random predictions for CIFAR-10 being expected to be around that accuracy.

3.2 Static Quantization (PTQ)

Static quantization utilizes PyTorch's FX Graph Mode to achieve a more aggressive form of compression. Unlike Dynamic quantization, static quantization transforms both the model weights and activations to int8 but requires a calibration phase. This phase is where a small representative dataset referred to as the "calibration data" is used to determine the optimal range and distribution of the activation values. This allows for efficient mapping of the floating point values to int8, without great accuracy loss.

This method produced a great reduction in model size, dropping down to **11.31MB**, inference latency reduced to **31.596 ms;** and accuracy still maintained steadily at **91.80%** a slight reduction but still great considering the reduced computational cost, as can be seen in the image below:

Figure 5: ResNet18 Static PTQ

On VGGNet, compared to the baseline of **90.73%** test accuracy with a model size of **537.31MB** and a latency of **355.77 ms,** the PTQ model reduced the size to **134.57MB** and had a similar accuracy of **90.90%**. The latency also improved to 31.596 **ms,** which is a marked improvement

over the baseline. The static PTQ model preserved model performance while still reducing the size and time.

```
[Static PTQ]
Model Size:          134.57 MB
Accuracy:            90.90%
Latency (CPU, ms):   239.735
```

Figure 6: VGGNet Static PTQ

This is in line with expectations, as static quantization is more aggressive.

3.3 Quantization Aware Training (QAT)

QAT differs from the two PTQ methodologies above in that it introduces the quantization inside the training process itself. During this, it simulates low-precision behavior using "fake quantization" modules inserted in the forward pass. These modules mimic the effects of quantization by rounding the values to a discrete integer before immediately converting them back to floating point using a scale and zero point. This lets the model experience the kind of errors it would encounter during inference.

The above processes, however, mean that QAT requires more careful tuning, which we did with 3 epochs for QAT, leading to a smaller model size, quicker inference latency, and greater accuracy.

This can be seen in the images below, as we obtain an accuracy of **94.72%** on the QAT_int8, while the size is reduced to **11.31MB** and the inference latency is 34.752 ms.

```
QAT Epoch 1 | Loss: 0.1723 | Fake-Quant Val Acc: 94.00%
QAT Epoch 2 | Loss: 0.1171 | Fake-Quant Val Acc: 94.57%
QAT Epoch 3 | Loss: 0.0864 | Fake-Quant Val Acc: 94.57%
```

```
Saved INT8 QAT model → /content/drive/MyDrive/resnet18_cifar10_QAT_int8.pth

--- QAT INT8 Metrics ---
Model size: 11.31 MB
Accuracy:   94.72%
Latency:    34.752 ms
```

Figure 7: ResNet18 QAT

Application of QAT on VGGNet, however, proved very difficult, as it required running on the CPU and not the GPU, and would continuously run for 16+hours before the session would time out on Google Colab. The screenshot below shows QAT on VGGNet, and training the baseline itself before getting to the QAT training using CPU took about 11 hours just to get to 37% and would constantly crash due to long hours of running.

```
Using: cpu
100%|██████████| 170M/170M [00:13<00:00, 12.4MB/s]
[Dataset] Train=50000, Test=10000
Downloading: "https://download.pytorch.org/models/vgg16_bn-6c64b313.pth" to /root/.cache/torch/hub/checkpoints/vgg16_bn-6c64b313.pth
100%|██████████| 528M/528M [00:09<00:00, 58.9MB/s]

=== TRAINING BASELINE (FP32) ===
 37%|███▋      | 579/1563 [11:59:15<20:02:47, 73.34s/it]
```

If allowed to run, however, we expect the QAT model to have reduced size and improved latency.

3.4 Structured Pruning

Structured pruning works differently from quantization in that it removes entire channels from convolutional layers based on an "importance" criterion. We used the L1-norm of the weights as our criterion, removing weights and filters with small L1-norms, thereby reducing the model's computational complexity and size. Unlike unstructured pruning, which zeros individual weights, structured pruning preserves the regularity of the model, allowing for reductions in the inference latency of the model.

After pruning 30% of the model and fine-tuning the pruned model for 3 epochs, the ResNet-18 model retained high accuracy while reducing model size and improving speed significantly. This can be seen in the image below as we obtained an accuracy of **84.42%**, a reduced model size of **10.88MB,** and an inference latency of **21.206ms**. The reduced model through pruning can be seen as the number of parameters has also dropped to **2.7M** from **11.1M**.

```
--- Pruned Model Metrics ---
Total params:        2,704,554
Trainable params:    2,704,554
Model Size:          10.88 MB
Accuracy:            84.42%
Latency (CPU, ms):   21.206
```

Figure 8: ResNet18 Pruning

Starting from a baseline VGG16-BN model trained on CIFAR-10, we got a test accuracy of **79.45%**, with a model size of **106.25MB**. After pruning **30%** of the channels, we obtained a higher test accuracy of **84.26%** while the model size dropped to **63.08MB**. This shows that structured pruning can compress the model and even improve generalization, and it is smaller than the baseline as well.

```
===== Final pruned + fine-tuned summary =====
Params: 16.523 M (baseline 27.836 M)
Size: 63.08 MB (baseline 106.25 MB)
Test Acc: 84.26% (baseline 79.45%)
```

Figure 9: VGGNet Pruning

3.5 Pruning and Quantization

Based on the outputs above, the pruned model reduced both latency and size considerably, while Quantisation Aware training gave us the best performance-to-size ratio. Therefore, we decided to apply both in tandem to the fine-tuned model and evaluate the resulting metrics. We expected this compressed model to be not only the smallest but also for the QAT to help keep accuracy at a high enough level.

This was confirmed by the output seen below. The model was reduced to **2.79MB** in size, with reduced parameters similar to the pruned model, while still maintaining an **87.26%** accuracy and a latency of **13.560 ms,** outperforming the quantized and pruned models in speed and size while still maintaining a high accuracy.

```
--- Pruned + QAT INT8 Metrics ---
Total params:        2,704,554
Trainable params:    2,704,554
Model Size:          2.79 MB
Accuracy:            87.26%
Latency (CPU, ms):   13.560
```

Figure 10: ResNet18 QAT + Pruning

On VGGNet, we got down to **36.55MB** of data occupied on the disk, which is lower than the pruned data (63.08 MB) and the baseline. Despite the additional compression, the accuracy is good at **84.24%,** which is slightly worse than the pruned model (84.26%). This shows that pruning followed by dynamic int8 quantization achieves size reduction without losing much in predictive performance.

```
===== Pruned + Quantized summary (dynamic int8) =====
Params: 7.248 M (same as pruned model 16.523 M)
Size: 36.55 MB (pruned FP32 63.08 MB, baseline FP32 106.25 MB)
Test Acc: 84.24% (pruned FP32 84.26%, baseline FP32 79.45%)
```

Figure 11: VGGNet QAT + Pruning

4.0 Conclusion

The combined application of structured pruning and quantization-aware training provides a clear improvement in size and inference latency while still maintaining a high enough accuracy. Compared to the baseline model, the pruned and quantized model provided a **significant** reduction in size and a reduction in inference latency across both the ResNet and VGGNet models, highlighting the effectiveness of the methods applied. These results illustrate that careful

and the controlled application of both pruning and quantization can yield a highly efficient and low computational-cost neural network model for implementation on edge devices.

Summary Table

Model	Accuracy (%)	Size (MB)	Latency (ms)	Total params (M)	Trainable Params (M)
Baseline	92.49	44.81	83.343	11.1	10.5
Dynamic Quantisation	92.54	44.79	85.139	11.1	10.5
Static Quantisation	91.80	11.31	31.596	11.1	10.5
QAT int_8	94.72	11.31	34.752	11.1	10.5
Structured Pruning	84.42	10.88	21.206	2.7	2.7
QAT + Pruning	87.26	2.79	13.56	2.7	2.7

Figure 12: ResNet18 Summary Table

Model	Accuracy (%)	Size (MB)	Latency (ms)	Total params (M)	Trainable Params (M)
Baseline	90.73	537.31	355.76	134.3	134.3
Dynamic Quantisation	11.09	71.81	362.93	14.7	14.7
Static Quantisation	90.9	134.57	239.73	134.3	134.3
QAT	N/A	N/A	N/A	N/A	N/A
Pruning	84.26	63.08	239	16.5	16.5
Pruning + QAT	84.24	36.55	239	7.24	7.24

Figure 13: VGGNet Summary Table

References:

[1] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and Quantization for Deep Neural Network Acceleration: A Survey,” June 15, 2021, *arXiv*: arXiv:2101.09671. doi: 10.48550/arXiv.2101.09671.

[2] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” Feb. 15, 2016, *arXiv*: arXiv:1510.00149. doi: 10.48550/arXiv.1510.00149.

[3] X. Dong, J. Huang, Y. Yang, and S. Yan, “More is Less: A More Complicated Network with Less Inference Complexity,” May 15, 2017, *arXiv*: arXiv:1703.08651. doi: 10.48550/arXiv.1703.08651.