CS6886 Systems Engineering for DL

CS25D007 Rahul Anilkumar

Assignment 3

1 Training Baseline

Following steps were performed to set up MobileNetV2 trained on CIFAR-10 dataset

(a) Prepare CIFAR-10 with proper normalization and data augmentation; specify transforms.

Answer.

- Normalization: CIFAR-10 dataset from Pytorch was used with normalization values [1]: (0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261).
- Transformations:
  - RandomCrop and RandomHorizontalFlip data augmentation steps were applied [2].
  - From baseline repository, CIFAR10Policy (25 policies like invert, rotate, sharpness etc chosen at random) and Cutout were used.
- Following transforms were applied to Train and Test sets:

```
# Training transforms with data augmentation
    trans_t = transforms.Compose(
        [transforms.RandomCrop(32, padding=4),
         transforms.RandomHorizontalFlip(),
         CIFAR10Policy(),
         transforms.ToTensor(),
         transforms.Normalize((0.4914, 0.4822, 0.4465),
                              (0.247, 0.243, 0.261)),
         Cutout(n_holes=1,length=16)
        ])

# Test transforms (no augmentation)
    trans = transforms.Compose(
        [transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465),
                             (0.247, 0.243, 0.261))])
])
```

(b) Describe your MobileNet-v2 configuration (e.g., width multiplier, dropout, BN settings) and training strategy (optimizer, LR schedule, regularization, epochs, batch size).

Answer.

- MobileNetV2 configuration parameters used in final model [3]

Table 1: Parameters used in final model

Parameter	**Description**	**Value used**
`num_classes`	Number of classification classes (model was updated to 10 classes in code)	Default
`width_mult`	Width multiplier – this controls the number of filters in every layer, setting value <1.0 would help compress the model, at the cost of accuracy	1.0
`inverted_residual_setting`	Allows to tweak the network of structure in different layers, no custom setting is used	None
`round_nearest`	Round the number of channels in each layer to be a multiple of this number. Used default setting.	8
`block`	Module specifying inverted residual building block, None as inverted_residual_setting was not used	None
`norm_layer`	No normalization layer was used, default setting	None
`dropout`	Probability of Dropout	0.2

- <u>Training Strategy used</u>

  MobileNetV2 for CIFAR-10 was trained from scratch on an A5000 GPU. A test accuracy of **94.87%** was obtained after training for 300 epochs with following settings:

  (a) Optimizer: SGD was used with initial learning rate of 0.07, with 0.9 momentum and 5e-4 weight_decay. SGD with momentum is well suited for imaging tasks, along with weight decay would prevent MobileNetv2 from overfitting to a small dataset like CIFAR-10.

  (b) Learning Rate Schedule: Used CosineAnnealing Learning rate scheduler.

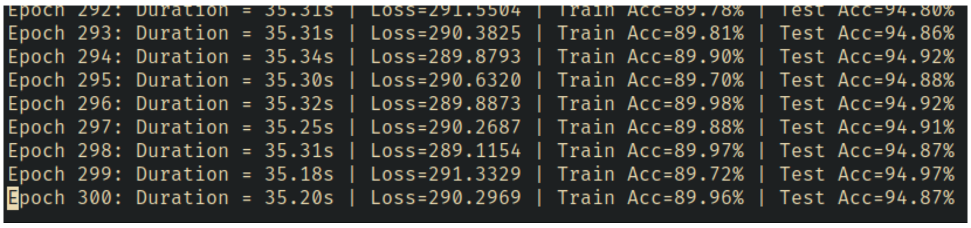
  (c) Regularization: Regularization via weight decay is used here.

  (d) Epochs: 300 epochs were chosen to get good accuracy since the training was from scratch

  (e) Batch Size: 128 was chosen as the batch size since the GPU could handle larger batches (initial trials on my laptop I had to resort to 32).

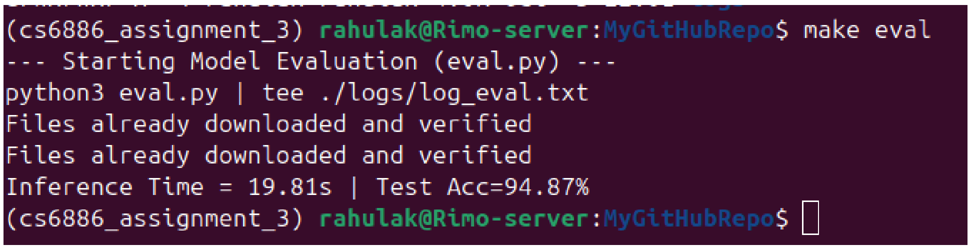
(c) Report final test top-1 accuracy and include loss/accuracy curves; briefly discuss failure modes.

Answer. The Final Test Top-1 accuracy obtained was **94.87%**.



```
Epoch 292: Duration = 35.31s | Loss=291.5504 | Train Acc=89.78% | Test Acc=94.80%
Epoch 293: Duration = 35.31s | Loss=290.3825 | Train Acc=89.81% | Test Acc=94.86%
Epoch 294: Duration = 35.34s | Loss=289.8793 | Train Acc=89.90% | Test Acc=94.92%
Epoch 295: Duration = 35.30s | Loss=290.6320 | Train Acc=89.70% | Test Acc=94.88%
Epoch 296: Duration = 35.32s | Loss=289.8873 | Train Acc=89.98% | Test Acc=94.92%
Epoch 297: Duration = 35.25s | Loss=290.2687 | Train Acc=89.88% | Test Acc=94.91%
Epoch 298: Duration = 35.31s | Loss=289.1154 | Train Acc=89.97% | Test Acc=94.87%
Epoch 299: Duration = 35.18s | Loss=291.3329 | Train Acc=89.72% | Test Acc=94.97%
Epoch 300: Duration = 35.20s | Loss=290.2969 | Train Acc=89.96% | Test Acc=94.87%
```

Figure 1: Training Log 94.87% Test Accuracy



```
(cs6886_assignment_3) rahulak@Rimo-server:MyGitHubRepo$ make eval
--- Starting Model Evaluation (eval.py) ---
python3 eval.py | tee ./logs/log_eval.txt
Files already downloaded and verified
Files already downloaded and verified
Inference Time = 19.81s | Test Acc=94.87%
(cs6886_assignment_3) rahulak@Rimo-server:MyGitHubRepo$
```

Figure 2: Evaluation of Trained Model

Failure to obtain good accuracy may be attributed to following reasons:

- MobileNet overfitting to CIFAR-10: MobileNet is built to handle larger images from ImageNet, and is prone to overfitting on the smaller CIFAR-10. To tackle this, regularization via weight decay is done, and transformations are applied to test set images to enhance generalization.
- Different input sizes and further resolution: Since ImageNet resolutions are large (224x224), MobileNet downsamples them with strides of 2, which might be detrimental to smaller image data from CIFAR-10 (32x32). To counter this, the strides of the first few layers were limited to 1 [4].
- Improper hyperparameters: Choosing the right number of epochs and learning rates did help get good accuracy. Lower number of epochs did result in poor accuracy, and lower learning rates were making training too slow. 300 epochs and 0.07 learning rate hit the sweet spot with training not taking much time, while maintaining accuracy. The use of SGD with CosineAnnealing was also a right choice for this imaging based task.

Train/Test Accuracy and Loss curves are below:

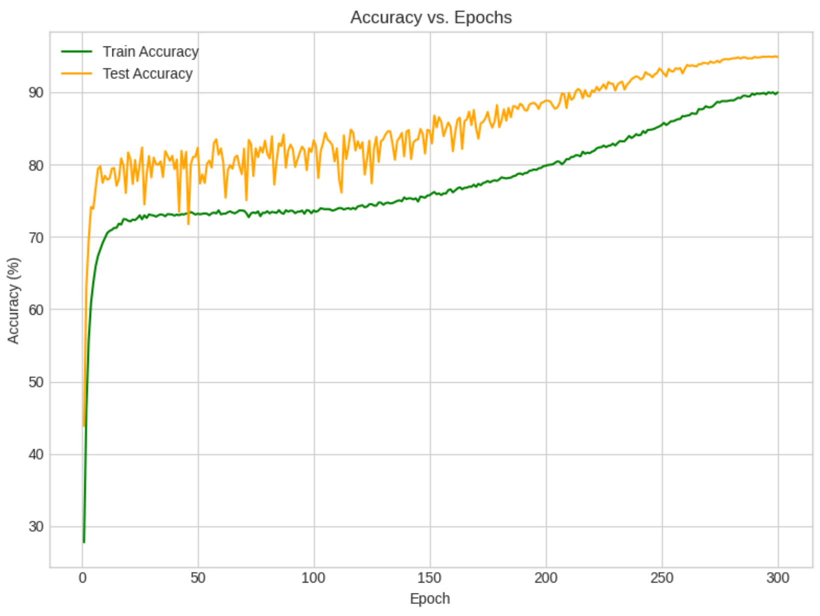


Figure 3: Accuracy vs Epochs

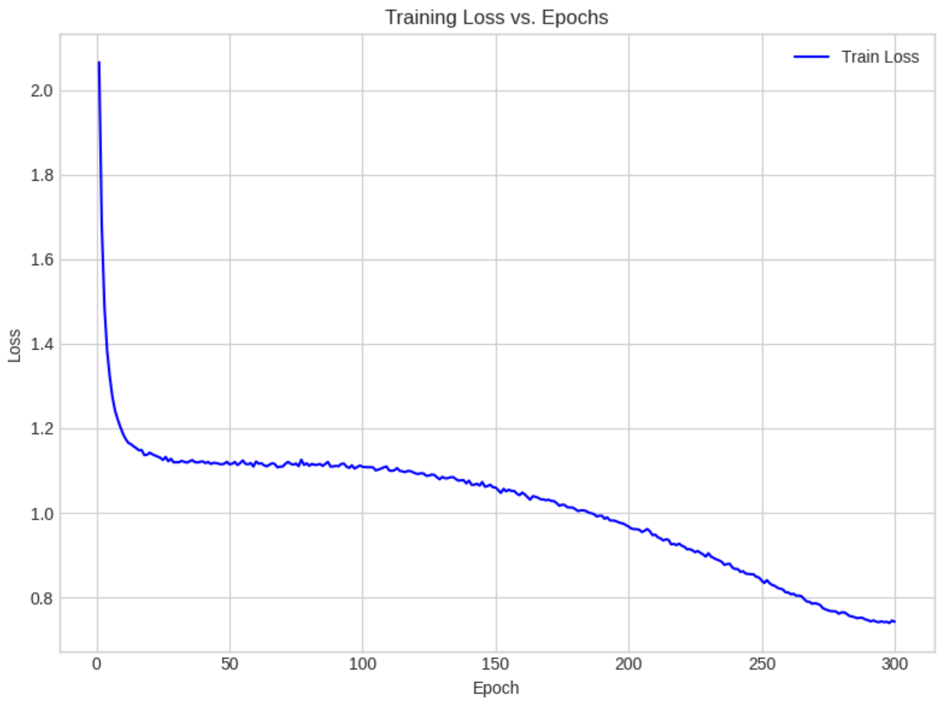


Figure 4: Training Loss vs Epochs

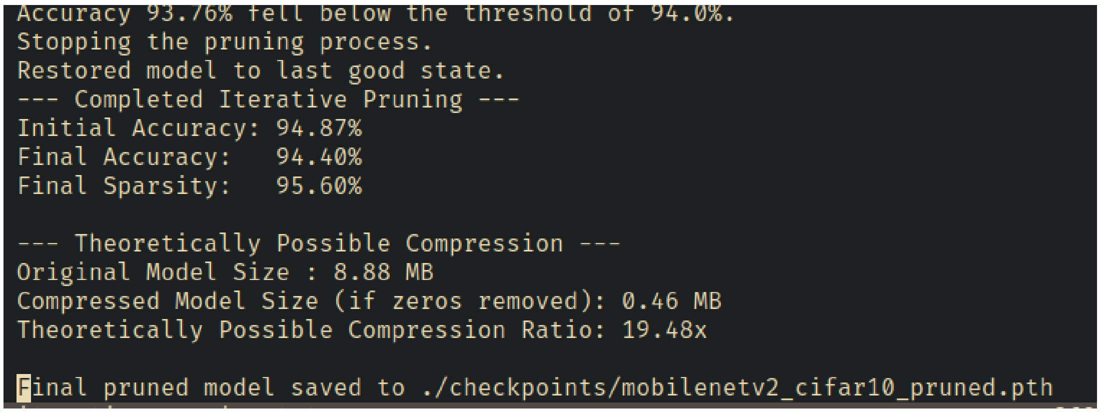
2 Model Compression Implementation

(a) Implement a configurable compression method for reducing both model weights and activations. Clearly explain your design choices.

Answer. The fundamental idea for the compression pipeline was inspired by the paper on Deep Compression [5]. Following are the stages implemented, along with the design choices:

2.1 Iterative Global Unstructured Magnitude Pruning with Fine Tuning

I was able to prune the model to a final sparsity of 95.60% and a test accuracy of **94.40%** (negligible drop from original 94.87%).



```
Accuracy 93.76% fell below the threshold of 94.0%.
Stopping the pruning process.
Restored model to last good state.
--- Completed Iterative Pruning ---
Initial Accuracy: 94.87%
Final Accuracy:   94.40%
Final Sparsity:   95.60%

--- Theoretically Possible Compression ---
Original Model Size : 8.88 MB
Compressed Model Size (if zeros removed): 0.46 MB
Theoretically Possible Compression Ratio: 19.48x

Final pruned model saved to ./checkpoints/mobilenetv2_cifar10_pruned.pth
```

Figure 5: Pruning Logs

The following design choices were made to arrive at a good accuracy rate:

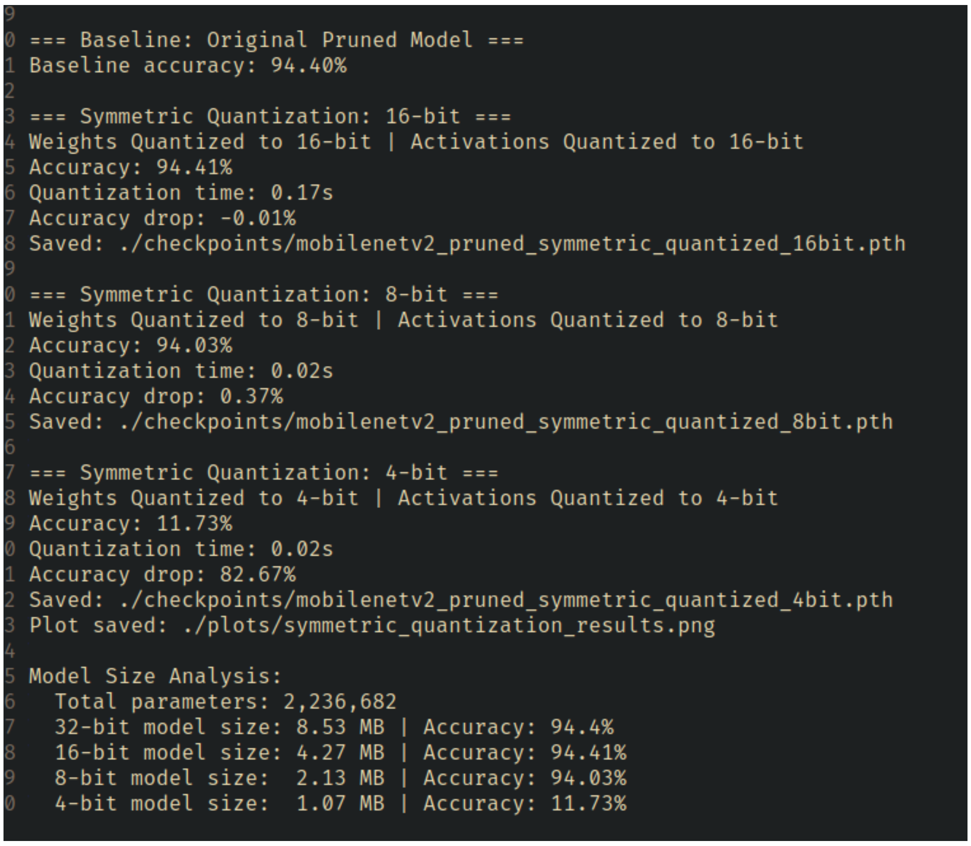
- Choice of Magnitude Pruning: Magnitude pruning post training was chosen since it's intuitive, and easier to implement. Initial runs gave good sparsity without significant drop in accuracy. The pruning was applied globally with a prune ratio of 20%.
- Iterative Pruning: The pruning was run iteratively, taking away 20% of the remaining weights every time. This will ensure a slowly degrading number of elements pruned away in each iteration.
- Fine Tuning: The pruned model was further fine tuned to improve accuracy, while maintaining the same sparsity (weights made zero will remain zero). The accuracy was evaluated against a threshold (94%) and if the model goes below this, the last good model is saved. A smaller learning rate (0.001) was used for fine tuning to stay close to the original accuracy.
- Choice of layers: Only Conv2D and Linear layers were pruned, BatchNorm and biases were not pruned (because they are small in size and might affect model stability). It is usual practice to not prune the first and last layers, but here the script was free to choose all Conv and linear layers. Since the accuracy did not degrade significantly, all layers were exposed to pruning.
- Experiments: Because many natural phenomenon follow Gaussian distributions, I did try out an experimental pruning technique termed Gaussian pruning, where in the weights are pruned from beginning to end following a Gaussian distribution (low pruning at beginning and end, peaking towards the middle). However, this did not yield good accuracy, so I dropped the idea.

2.2 Symmetric Linear Quantization with Zero-Preservation

Quantization was applied to Conv2D and linear layer weights for different bit widths (16/8/4). A configurable quantization script is added here to choose between (2,4,6,8,12,16,32) bits.

- Weight Quantization: Since weights are fixed after training (and pruning), a static quantization was applied (quantized once when converting the model). Scale and shift technique discussed in the lectures were used to achieve this.
- Activation Quantization: As activations change during run time depending on input, a dynamic quantization was done during each forward pass.
- Zero Preservation: The pruned model had good amount of sparsity, to make use of that sparsity, it was decided to keep the zeros same for quantized weights as well.

The accuracies and theoretical weight reductions for different bit widths (only listed values where weights and activations use same bit width):

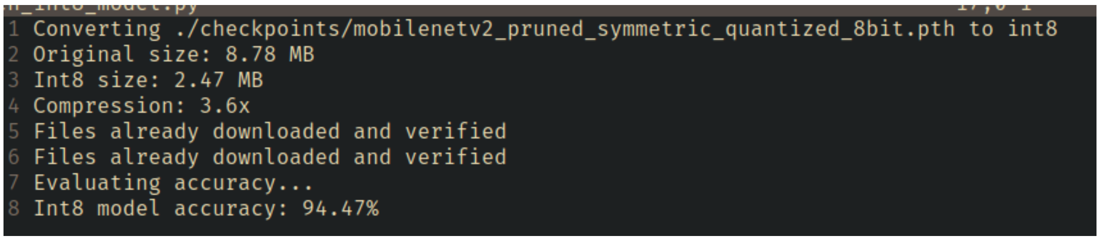


```
=== Baseline: Original Pruned Model ===
Baseline accuracy: 94.40%

=== Symmetric Quantization: 16-bit ===
Weights Quantized to 16-bit | Activations Quantized to 16-bit
Accuracy: 94.41%
Quantization time: 0.17s
Accuracy drop: -0.01%
Saved: ./checkpoints/mobilenetv2_pruned_symmetric_quantized_16bit.pth

=== Symmetric Quantization: 8-bit ===
Weights Quantized to 8-bit | Activations Quantized to 8-bit
Accuracy: 94.03%
Quantization time: 0.02s
Accuracy drop: 0.37%
Saved: ./checkpoints/mobilenetv2_pruned_symmetric_quantized_8bit.pth

=== Symmetric Quantization: 4-bit ===
Weights Quantized to 4-bit | Activations Quantized to 4-bit
Accuracy: 11.73%
Quantization time: 0.02s
Accuracy drop: 82.67%
Saved: ./checkpoints/mobilenetv2_pruned_symmetric_quantized_4bit.pth
Plot saved: ./plots/symmetric_quantization_results.png

Model Size Analysis:
  Total parameters: 2,236,682
  32-bit model size: 8.53 MB | Accuracy: 94.4%
  16-bit model size: 4.27 MB | Accuracy: 94.41%
  8-bit model size:  2.13 MB | Accuracy: 94.03%
  4-bit model size:  1.07 MB | Accuracy: 11.73%
```

Figure 6: Quantization Logs

2.3 Showing Actual Size reduction: Generate int8 model

The 8-bit model showed the best tradeoff between accuracy and size after quantization, therefore it was converted to int8 (from float32) to show actual model size reduction. The int8 model had a test accuracy of 94.47%.



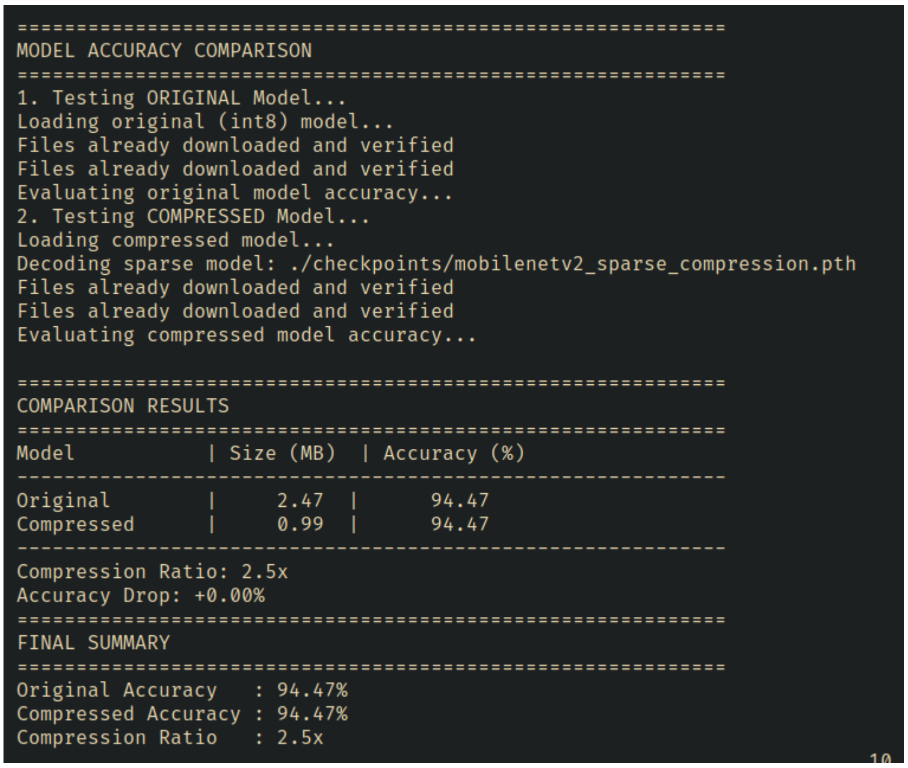
```
Converting ./checkpoints/mobilenetv2_pruned_symmetric_quantized_8bit.pth to int8
Original size: 8.78 MB
Int8 size: 2.47 MB
Compression: 3.6x
Files already downloaded and verified
Files already downloaded and verified
Evaluating accuracy...
Int8 model accuracy: 94.47%
```

Figure 7: Model Size reduced to 2.47MB after int8 conversion

2.4 Sparse Compression - Because Theoretical Savings aren't enough

The model, even though sparse, is still storing zero weights as float32/int8 values, so the model size does not actually come down. The Deep Compression paper [5] uses Huffman encoding (frequency based bit encoding for weights) to squeeze out maximum savings. Here, I decided to use a simple Sparse compression that stores indices (and values) of non-zero weights, instead of storing the zero weights. Design iterations:

- Initial compression took more space (3MB against 2MB) because indices were stored as int64
- Updated code to save as int8/16/32 where possible.
- Finally model came down to $\approx$ 1MB with no accuracy loss from original

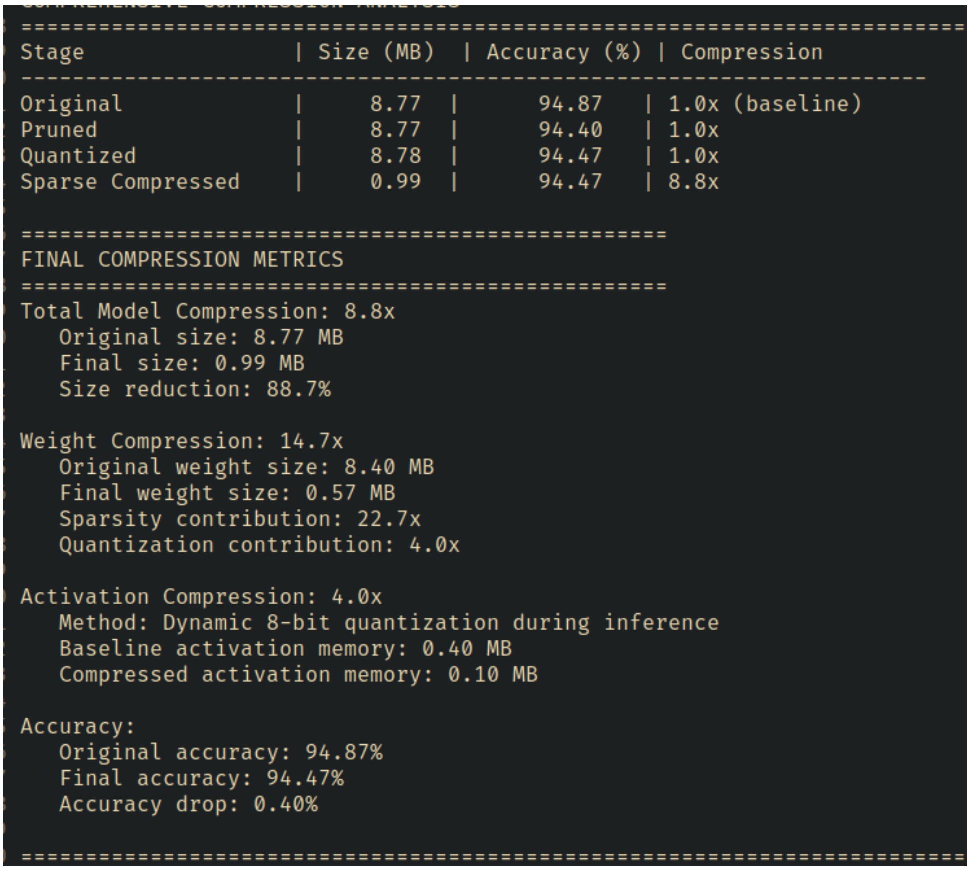


```
============================================================
MODEL ACCURACY COMPARISON
============================================================
1. Testing ORIGINAL Model...
Loading original (int8) model...
Files already downloaded and verified
Files already downloaded and verified
Evaluating original model accuracy...
2. Testing COMPRESSED Model...
Loading compressed model...
Decoding sparse model: ./checkpoints/mobilenetv2_sparse_compression.pth
Files already downloaded and verified
Files already downloaded and verified
Evaluating compressed model accuracy...

============================================================
COMPARISON RESULTS
============================================================
Model           | Size (MB)  | Accuracy (%)
------------------------------------------------------------
Original        |     2.47  |      94.47
Compressed      |     0.99  |      94.47
------------------------------------------------------------
Compression Ratio: 2.5x
Accuracy Drop: +0.00%
============================================================
FINAL SUMMARY
============================================================
Original Accuracy   : 94.47%
Compressed Accuracy : 94.47%
Compression Ratio   : 2.5x
```

Figure 8: Sparse Compression Logs

(b) Show how the compression is applied to MobileNet-v2 (which layers compressed, any exceptions).

Answer. The compression pipeline was able to achieve a compression ratio of **8.8x** (Actual model size reduction from 8.77 MB to just 0.99 MB) with minimal drop in accuracy (**0.40%**).



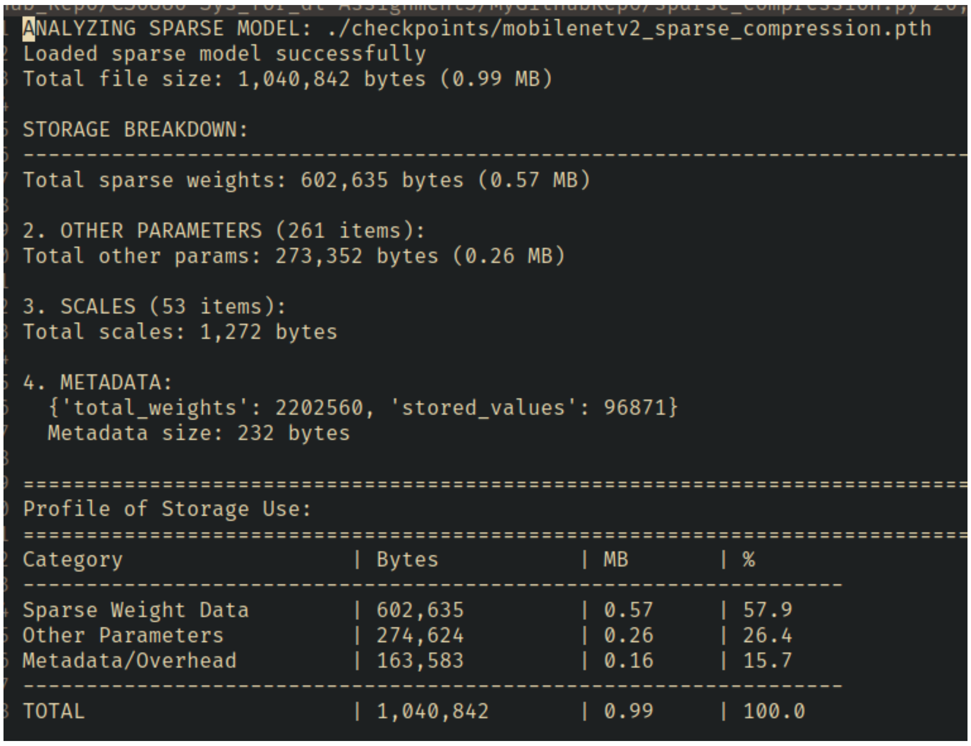
```
=====================================================================
Stage                  | Size (MB)  | Accuracy (%) | Compression
------------------------------------------------------------------
Original               |     8.77   |      94.87   | 1.0x (baseline)
Pruned                 |     8.77   |      94.40   | 1.0x
Quantized              |     8.78   |      94.47   | 1.0x
Sparse Compressed      |     0.99   |      94.47   | 8.8x

==================================================
FINAL COMPRESSION METRICS
==================================================
Total Model Compression: 8.8x
   Original size: 8.77 MB
   Final size: 0.99 MB
   Size reduction: 88.7%

Weight Compression: 14.7x
   Original weight size: 8.40 MB
   Final weight size: 0.57 MB
   Sparsity contribution: 22.7x
   Quantization contribution: 4.0x

Activation Compression: 4.0x
   Method: Dynamic 8-bit quantization during inference
   Baseline activation memory: 0.40 MB
   Compressed activation memory: 0.10 MB

Accuracy:
   Original accuracy: 94.87%
   Final accuracy: 94.47%
   Accuracy drop: 0.40%

=====================================================================
```

Figure 9: Final Compression Metrics

The details of layers compressed, exceptions are listed as part of (a), mentioning here: Choice of layers: Only Conv2D and Linear layers were pruned, BatchNorm and biases were not pruned (because they are small in size). It is usual practice to not prune the first and last layers, but here the script was free to choose all conv and linear layers. Since the accuracy did not degrade significantly, all layers were exposed to pruning.

(c) Document storage overheads (e.g., metadata, scaling factors) and include them in size estimates.

Answer. The overheads in this case would come from storing the indices for non-zero values, scales from quantization etc. The split-up of size used be weights, other parameters (bias, batchnorm), and metadata (scales/indices) are shown below:



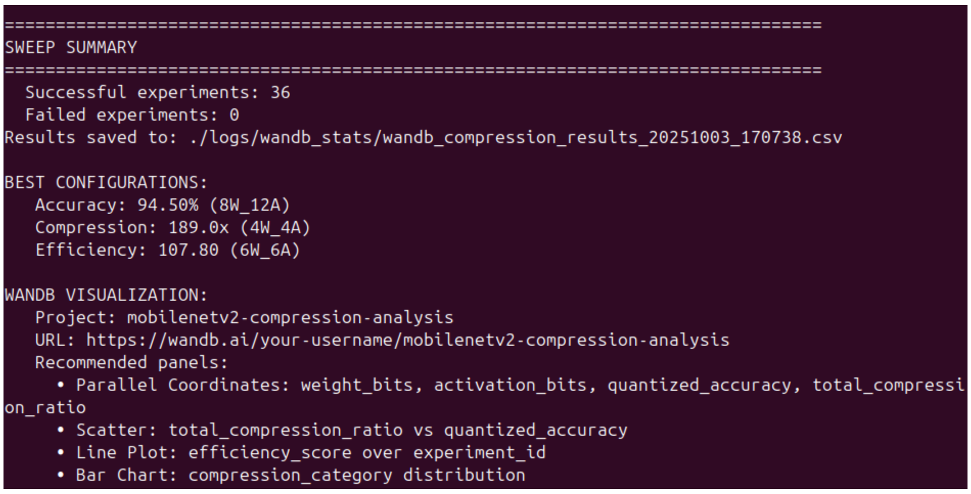
```
ANALYZING SPARSE MODEL: ./checkpoints/mobilenetv2_sparse_compression.pth
Loaded sparse model successfully
Total file size: 1,040,842 bytes (0.99 MB)

STORAGE BREAKDOWN:
------------------------------------------------------------------------------
Total sparse weights: 602,635 bytes (0.57 MB)

2. OTHER PARAMETERS (261 items):
Total other params: 273,352 bytes (0.26 MB)

3. SCALES (53 items):
Total scales: 1,272 bytes

4. METADATA:
  {'total_weights': 2202560, 'stored_values': 96871}
  Metadata size: 232 bytes

==============================================================================
Profile of Storage Use:
==============================================================================
Category                  | Bytes            | MB         | %
------------------------------------------------------------------
Sparse Weight Data        | 602,635          | 0.57       | 57.9
Other Parameters          | 274,624          | 0.26       | 26.4
Metadata/Overhead         | 163,583          | 0.16       | 15.7
------------------------------------------------------------------
TOTAL                     | 1,040,842        | 0.99       | 100.0
```

Figure 10: Data Storage Profile

3 Compression Results

(a) Apply your compression pipeline at different levels of compression (e.g., varying bit-widths or parameters).

Answer. The compression pipeline was applied with 36 different combinations of bit-widths of weights and activations, and the results captured by wandb plot. A summary of the sweep results is shown here (complete logs are captured as csv file in GitHub repo):



```
================================================================================
SWEEP SUMMARY
================================================================================
  Successful experiments: 36
  Failed experiments: 0
Results saved to: ./logs/wandb_stats/wandb_compression_results_20251003_170738.csv

BEST CONFIGURATIONS:
   Accuracy: 94.50% (8W_12A)
   Compression: 189.0x (4W_4A)
   Efficiency: 107.80 (6W_6A)

WANDB VISUALIZATION:
   Project: mobilenetv2-compression-analysis
   URL: https://wandb.ai/your-username/mobilenetv2-compression-analysis
   Recommended panels:
     • Parallel Coordinates: weight_bits, activation_bits, quantized_accuracy, total_compressi
on_ratio
     • Scatter: total_compression_ratio vs quantized_accuracy
     • Line Plot: efficiency_score over experiment_id
     • Bar Chart: compression_category distribution
```

Figure 11: Compression Pipeline sweep with different bit-widths

(b) Evaluate and compare the accuracy in these settings. Provide the Wandb Parallel Coordinates chart.

Answer. The accuracy and compression details of various settings is listed below:

Experiment_No	**bit_config**	**theoretical_size_mb**	**quantized_accuracy**
1	32W_32A	0.684465	94.40
2	32W_16A	0.684465	94.40
3	32W_12A	0.684465	94.43
4	32W_8A	0.684465	94.14
5	32W_6A	0.684465	91.21
6	32W_4A	0.684465	17.70
7	16W_32A	0.499699	94.40
8	16W_16A	0.499699	94.41
9	16W_12A	0.499699	94.41
10	16W_8A	0.499699	94.26
11	16W_6A	0.499699	91.28
12	16W_4A	0.499699	18.48
13	12W_32A	0.453507	94.42
14	12W_16A	0.453507	94.41
15	12W_12A	0.453507	94.43
16	12W_8A	0.453507	93.93
17	12W_6A	0.453507	91.22
18	12W_4A	0.453507	18.78
19	8W_32A	0.407315	94.47
20	8W_16A	0.407315	94.47
21	8W_12A	0.407315	**94.50**
22	8W_8A	0.407315	94.03
23	8W_6A	0.407315	91.31
24	8W_4A	0.407315	17.36
25	6W_32A	0.384219	94.19
26	6W_16A	0.384219	94.19
27	6W_12A	0.384219	94.20
28	6W_8A	0.384219	93.84
29	6W_6A	0.384219	91.02
30	6W_4A	0.384219	17.29
31	4W_32A	0.361124	72.54
32	4W_16A	0.361124	72.55
33	4W_12A	0.361124	72.52
34	4W_8A	0.361124	72.62
35	4W_6A	0.361124	68.28
36	4W_4A	0.361124	11.73

Table 2: Compression experiments with different bit configurations

The Wandb Parallel Coordinates chart is given below:

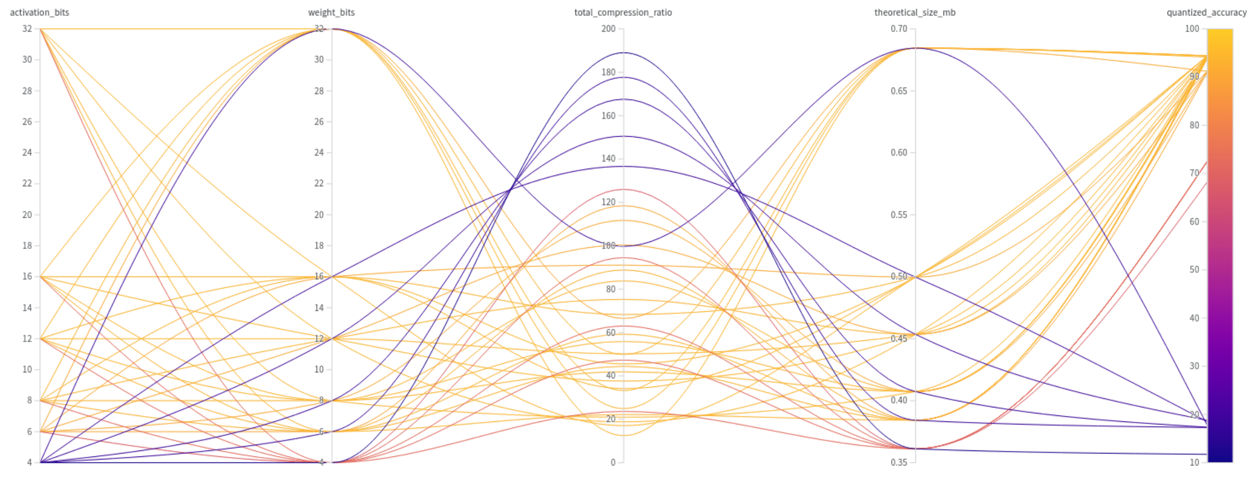


Figure 12: Wandb Parallel Coordinates Chart for MobileNetV2 compression

4 Compression Analysis

Figure-9 gives the values from the log files, which are reproduced here.

(a) Compression ratio of the model

Answer. Final Compression Ratio of the Model = **8.8x**

(b) Compression ratio of the weights

Answer. Final Compression Ratio of the Weights = 14.7x

(c) Compression ratio of the activations

Answer. Final Compression Ratio of the Activations = 4.0x
The activation compression ratio is measured by comparing original (float32) bit-width to the quantized bit-width (int8 in this case), as all activations are dynamically quantized to this width.

(d) Final approximated model size (MB) after compression.

Answer. Final Actual Model Size (MB) after compression = **0.99 MB** (down from 8.77 MB)

5 Reproducibility Repository

(a) Clean, modular,well commented codebase with separation of training, evaluation, and compression.

Answer. The code is well structured with a proper Makefile for running different steps for training, evaluation and compression.

(b) README with exact commands, environment, and dependency versions; include seed configuration.

Answer. README.md file with the exact commands and seed configuration is added. Environment and dependency versions are captured as a requirements.txt file in the repository. Python version: 3.10.12 PCs without torch+cu might need to install pytorch separately with: pip3 install torch torchvision torchaudio

(c) Provide the GitHub repository link.
Answer. https://github.com/rahulak-cs-iitm/cs6886-jul-nov-2025-assignment-3.git

References

[1] D. Macêdo, "Correct Normalization values for CIFAR-10." Accessed: 25-09-2025.

[2] PyTorch, "PyTorch - Transforming images, videos, boxes and more." Accessed: 25-09-2025.

[3] Pytorch, "MobileNetV2 in Pytorch." Accessed: 26-09-2025.

[4] chenghan98, "mobileNet-v2_cifar10." Accessed: 03-10-2025.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.