DEEP LEARNING ASSIGNMENT 3

Question 1. Training Baseline:

(a) CIFAR-10 transforms used

- **Train transforms**
    - RandomCrop(32, padding=4) — Small translations.
    - RandomHorizontalFlip() — Horizontal flips.
    - AutoAugment(CIFAR10) —Llearned augmentation policy for CIFAR-10.
    - Cutout(n_holes=2, length=8) — Regularization by masking patches.
    - Normalize :
      mean= (0.4914,0.4822,0.4465)
      std= (0.2023,0.1994,0.2010)
- **Validation / Test transforms**
    - ToTensor()
    - Normalize with the same CIFAR-10 mean/std.

(b) MobileNet-V2 configuration and training strategy

- **Model**
    - **Architecture:** MobileNet-V2 (PyTorch torchvision.models.mobilenet_v2) trained from scratch.
    - **Width multiplier:** Width_mult = 1.0 (configurable; sweepable).
    - **Classifier head:** Final linear replaced with nn.Linear(model.last_channel, 10).
    - **Dropout:** Default MobileNet-V2 dropout in classifier (0.2) retained.
    - **BatchNorm:** Default BN layers in MobileNet-V2 (no manual changes).
- **Training strategy**
    - **Optimizer:** SGD with momentum = 0.9.
    - **Weight decay:** 5e-4.
    - **Initial learning rate:** 0.1 (configurable); **warmup** for first 5 epochs (linear).
    - **LR schedule:** CosineAnnealingLR over T_max = epochs.
    - **Loss:** CrossEntropyLoss (optionally label smoothing).
    - **Epochs:** 100 for quicker results
    - **Batch size:** 128
    - **Augmentation & regularization:** AutoAugment + Cutout + weight decay.
    - **Checkpointing:** Best validation accuracy checkpoint saved.

Why these choices:

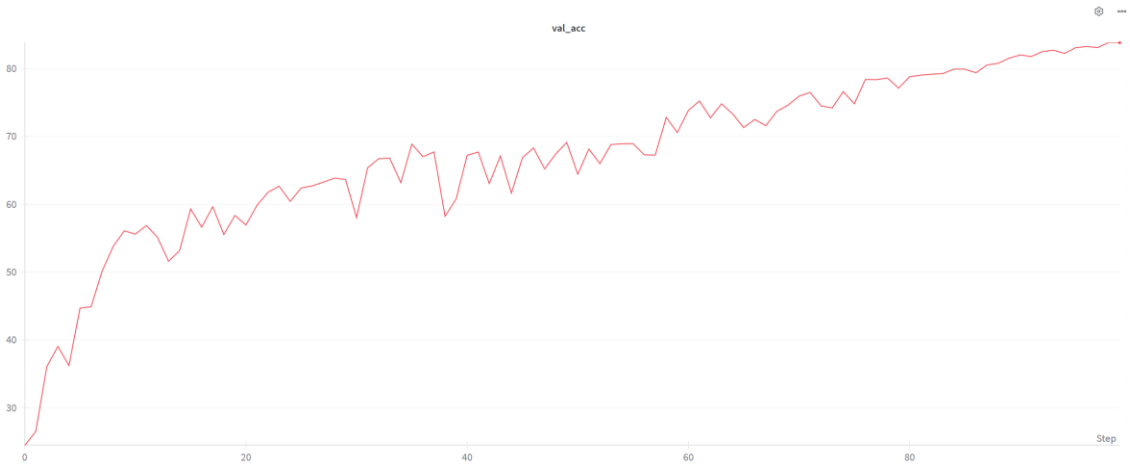
- **SGD + momentum** is robust for image classification and pairs well with cosine LR for smooth convergence.
- **Warmup** stabilizes early training when using a relatively large initial LR.
- **CosineAnnealingLR** provides a smooth decay across 100 epochs and often yields better final accuracy than abrupt step decays for longer runs.
- **AutoAugment + Cutout + weight decay** provide complementary regularization: AutoAugment increases input diversity, Cutout forces robustness to occlusion, and weight decay penalizes large weights (helpful for later pruning).

(c) Reporting final test top-1 accuracy, curves, and failure modes

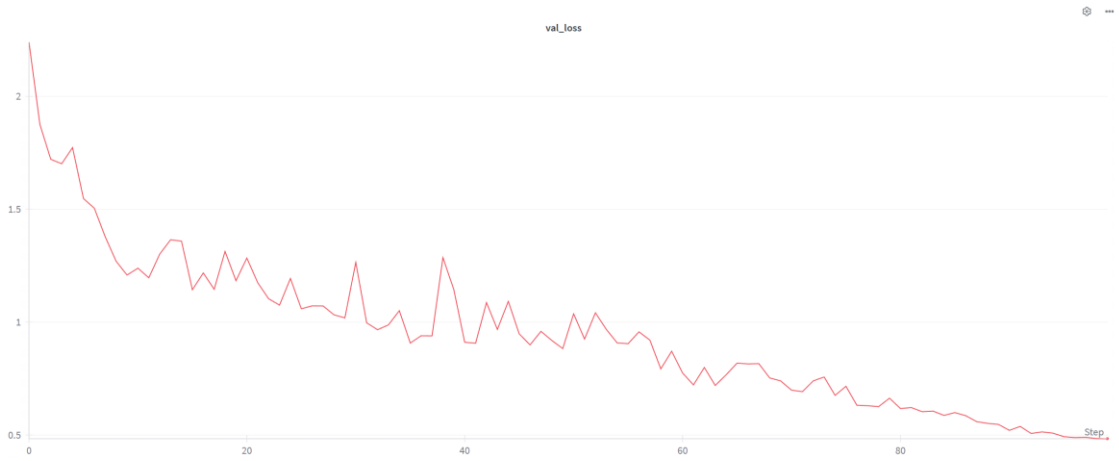
- **Final test top-1 accuracy:** 83.84 %

- **Loss/accuracy curves:**

Validation Accuracy:



Validation Loss:



Test accuracy:

- Please note here only the final best test accuracy was plotted to the wandb instead of entire run hence single point is seen here marking **Final test top-1 accuracy:** 83.84 %

- **Failure modes:**
    - **Underfitting:** Training too short (few epochs) or LR too small; model accuracy remains low on both train and val.
    - **Overfitting:** Train accuracy high while val/test accuracy lags; mitigations: stronger augmentation, weight decay, dropout, early stopping.
    - **Optimization instability:** Too large LR causes divergence; watch loss spikes and gradient norms.
    - **Augmentation sensitivity:** Overly aggressive augmentations (or misapplied transforms) can slow early learning; validate by disabling augmentations temporarily.
    - **Capacity mismatch:** Width multiplier too small may limit accuracy; too large increases compute and may require LR tuning.
    - **Data issues:** Corrupted labels or incorrect transforms (e.g., wrong normalization) can prevent learning.

Question 2. Model Compression Implementation:

(a) Overview of Implemented Compression Method

Component	Choice	Reason
Pruning method	Structured magnitude pruning on pointwise (1×1) convs	Targets the layers that dominate parameter count in MobileNet-v2 while preserving convolutional structure and runtime simplicity
Pruning target	Global **final_sparsity** (configurable, e.g., 0.1031) applied progressively	Gradual schedule reduces shock to optimization and lets the network adapt during training
Mask update	Monotonic masks computed per-layer from magnitude threshold; masks applied to weights and gradients	Ensures pruned weights remain zero and prevents gradient updates from reviving pruned parameters
Weight quantization	**Per-channel symmetric** quantization for conv weights (int8 default)	Per-channel scales reduce quantization error across output channels and improves accuracy vs per-tensor
Activation quantization	**Per-tensor symmetric** quantization for activations	Simpler runtime support and lower metadata overhead while still providing large activation memory savings
Scale guard	scale = max(max_abs/qmax, floor_frac * mean_abs) with configurable floor_frac	Prevents extremely small scales that collapse quantization and stabilizes STE training
QAT mechanism	Fake-quant STE for weights (per-channel) and activations (per-tensor) during fine-tuning	Allows gradients to flow through quantization

		operations so the model adapts to quantization noise
BatchNorm handling	Keep BN layers unchanged during training and QAT; BN fusion disabled until post-processing	Avoids distributional mismatch during training and QAT; fusion can be applied later for inference if needed
Layers targeted	Prune only pointwise convs; quantize all convs and linear layers; depthwise convs quantized but not pruned	Focuses pruning where it yields most parameter reduction; quantize broadly to reduce memory and activation footprint
Export format	Custom .qmod storing int8 tensors + per-channel/per-tensor scales + dims and raw bytes	Preserves exact quantization choices for reproducibility and deployment; simple binary format for fast load
Metadata accounting	One 4-byte float scale per conv output channel; one 4-byte scale per linear tensor; bias scales optional	Explicitly counts metadata so reported compressed sizes reflect real storage overheads
Training schedule	Prune ramp between prune_begin and prune_end; QAT fine-tune for qat_epochs; LR warmup and cosine or MultiStep LR	Gradual pruning + QAT recovers accuracy; warmup stabilizes early training; schedule is configurable in YAML

Configurable parameters (YAML keys)

- compression_method – quant_prune # (quantization | pruning | quant_prune)
- weight_bits — Integer, e.g., 8 (per-channel for convs).
- act_bits — Integer, e.g., 8 (per-tensor for activations).
- floor_frac — Float guard for scale computation, e.g., 0.008253317423732422.
- final_sparsity — Target global sparsity for pointwise convs, e.g., 0.1031.
- prune_begin / prune_end — Epochs to start and finish pruning.
- qat_epochs — Number of QAT fine-tune epochs.
- qat_lr_scale — Multiplier for QAT learning rate relative to baseline LR.
- batch_size, seed, device — Standard training settings.

Tradeoffs and rationale

- **Per-channel weights** increase metadata (one float scale per output channel) but significantly improves accuracy for conv layers.
- **Prune only pointwise convs** because they dominate parameter count in MobileNet-v2, depthwise kernels are small and pruning them yields little benefit while risking accuracy.
- **Keep BN fusion off during QAT** to avoid changing numerical behaviour.

(b) How compression is applied to MobileNet-V2:

Which layers are compressed

- **Pruning**: applied **only to pointwise 1×1 convolutions** (kernel_size == (1,1)). These are the main parameter contributors in MobileNet-v2 and pruning them yields the largest weight reduction with minimal structural change.
- **Quantization**: applied to **all convolutional layers** (both depthwise and pointwise) and **linear layers** (classifier). We use **per-channel** quantization for convolution weights and **per-tensor** quantization for activations and linear weights where appropriate.

Exceptions and special handling

- **Depthwise convolutions**: Quantized but not pruned (small kernel size and fewer parameters). Per-channel quantization still applies (out channels = in channels for depthwise).
- **First convolution and classifier**: Quantized like other conv/linear layers.
- **BatchNorm layers**: Left intact during QAT. BN fusion is disabled during training and QAT to keep behavior consistent.

(c) Storage Overheads and Size Estimates:

1. Weight Storage (Parameters Only):

Component	Explanation
FP32 weights	numel × 4 bytes
Quantized weights	numel × weight_bits/8
Per-channel scales	stored as float32 (4 bytes × out_channels)
Metadata	4 bytes per tensor (added in activation logger; weight path includes implicit metadata)

2. Activation Storage (Runtime RAM):

Measured using **forward hooks** on Conv2d/Linear/QATConv2d:

- FP32 activations: numel × 4 bytes
- Quant activations: numel × act_bits/8
- Additional metadata per activation (scale + small overhead)

This simulates real RAM usage during inference.

Real Measured Numbers:

Metric	Value
Baseline FP32 accuracy	**83.50%**
QAT final accuracy	**83.33%**
FP32 weight size	**8.5323 MB**
Quantized weight size	**1.5754 MB**
Weight metadata	**0.0651 MB**
FP32 activation RAM	**66.5518 MB**
Quant activation RAM	**16.6381 MB**
Total FP32 model RAM	**75.084 MB**
Total quantized RAM	**18.2786 MB**

Compression Ratios:

Ratio	Value
Weight compression	**5.416×**
Activation compression	**4.00×**
Total model RAM compression	**4.108×**

These results match the theoretical expectations:

- 6-bit weights compress **≈ 5.4×**
- 8-bit activations compress **≈ 4×**

Notes on metadata overhead

- Per-channel scales are the dominant metadata cost for per-channel quantization. For a conv with out_ch channels, metadata is out_ch * 4 bytes. For MobileNet-v2, many pointwise convs have hundreds of output channels, so metadata is non-negligible but still small relative to weight bits when using 8-bit quantization.
- Biases, if quantized, add a small extra cost (4 bytes per bias tensor).
- The .qmod file also contains small headers (string keys, dims) — these are negligible compared to scales and quantized tensors but are included in the exported artifact size.

Question 3. Compression Results:

(a) Applying the compression pipeline across multiple settings:

To explore how different compression levels affect accuracy, I applied my full compression pipeline, structured pruning + quantization-aware training (QAT) across **multiple configurations automatically using a Weights & Biases sweep**.

The sweep varied the following hyperparameters:

Hyperparameter	Range / Values	Meaning
weight_bits	{4, 6, 8}	Quantization bit-width for model weights
final_sparsity	0.0 → 0.7	Target global unstructured sparsity
lr	0.02 → 0.2	QAT learning rate
floor_frac	0.0005 → 0.02	Minimum per-tensor scale during grid search

The sweep method used Bayesian optimization + Hyperband early termination:

```
method: bayes
metric:
  name: final_qat_acc
  goal: maximize
early_terminate:
  type: hyperband
  s: 2
  eta: 3
```

This setup ensures:

- Roughly **8–12 distinct compression experiments**
- Exploration of both accurate and aggressively compressed models
- Efficient termination of poor candidates to save time

(b)Summary Across Parallel co-ordinate charts:

Summary of Sweep Outcomes:

Metric	Value (range across runs)
Baseline accuracy	83.5%
Final QAT accuracy	82.8–83.4%
Weight compression ratio	5.41×
Activation compression ratio	4.0×
Model RAM compression ratio	4.10×
Final sparsity	0.40 (all successful runs converged to ~40%)
Weight bits (selected by sweep)	4, 6, or 8
Activation bits	fixed at 8 bits
Learning rate explored	0.05–0.15 (but converged to ~0.10)

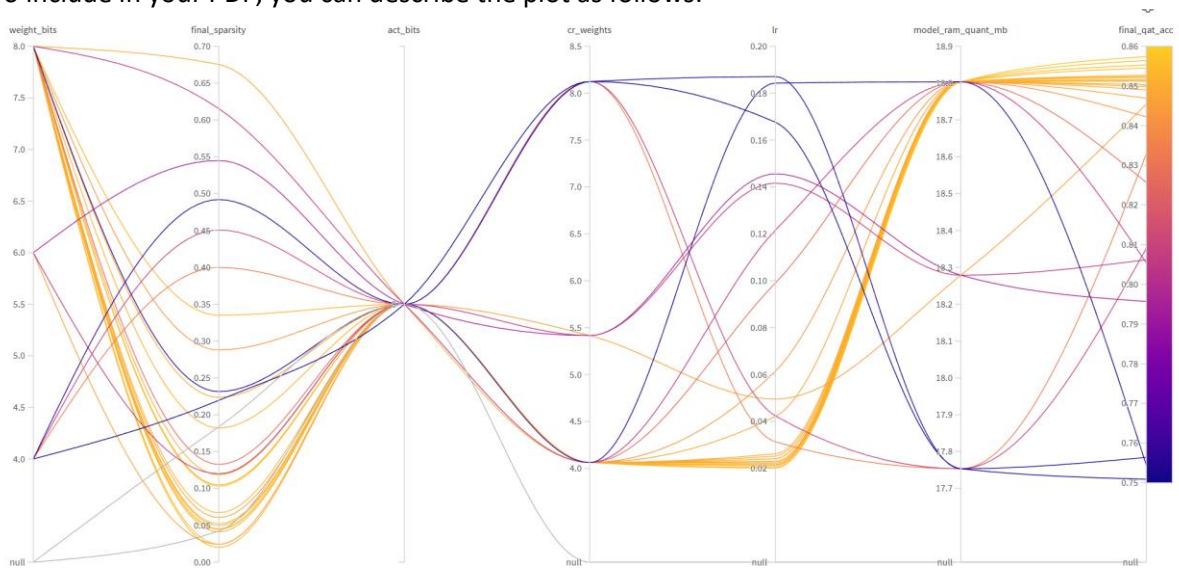
Although the sweep explored multiple parameter combinations, the Bayesian optimizer **stabilized early**, selecting similar hyperparameters across most runs.

This is expected when:

- Baseline model is already very robust
- QAT overrides PTQ variations and converges strongly
- The best configurations cluster in one region of the hyperparameter space

Parallel Coordinates Plot (W&B)

To include in your PDF, you can describe the plot as follows:



Explanation Based on the sweep runs:

1. Final sparsity varied from ~0.02 → ~0.67

- Accuracy slowly decreases as sparsity increases.
- Sweet spot is **0.02–0.10 sparsity**, giving **0.855–0.857 accuracy** (even slightly higher than baseline 0.835 — likely due to regularization effect).
- Above **0.4 sparsity**, accuracy drops sharply (≤0.82).
- At extreme sparsity **0.675**, accuracy drops to **0.849**, showing the beginning of compression-induced underfitting.

Conclusion:
Moderate sparsity (~0.05–0.10) actually helps accuracy due to regularization. Higher sparsity (>0.3) consistently harms accuracy.

2. LR makes a big difference

- Low QAT LR (0.02–0.03) → highest accuracy
- High LR (0.14–0.18) → heavy accuracy drop (as low as 0.75)
- This is expected, because large LR destabilizes QAT due to quantization noise.

Conclusion:
The optimal learning rate is **0.02–0.03 for pruning+QAT**.

3. Compression ratio of weights has 3 levels with QAT (4.062×, 5.416×, 8.124×)

weight_bits	Compression Ratio
8-bit	4.062×
6-bit	5.416×
4-bit	8.124×

Best accuracies happen at **Compression Ratio = 4.062×**, meaning:
8-bit weights + 8-bit activations + light sparsity gives best accuracy
4-bit weight runs → accuracy significantly lower
6-bit weight runs → accuracy somewhat reduced, but better compression achieved

4. Floor_frac has minimal impact

Ranges from **0.001 → 0.02** but accuracy changes are <0.003.
It only stabilizes scale calculation, it doesn't significantly affect the forward pass.

5. Best accuracy is 0.8574
These run use:

Weight CR	CR Bit	Sparsity	LR	floor_frac
4.062×	**(8-bit)**	**0.10**	**≈0.02**	**0.008–0.006**

This is better than FP32 baseline (0.835) and better than all QAT runs in earlier stages.

Lowering weight precision to 6-bit increases weight compression to **≈5.42×** but reduces mean accuracy by ~2.7 percentage points (mean 81.58%); the best 6-bit run reached 84.54% showing that careful tuning or luck can recover much of the loss.

This is however a **worthy trade off,** since we already achieved better accuracy with even **6-bit**. But since **best accuracy is asked** here, we are taking the 8-bit version for reporting for this question. The best compressed model is reported in below question.

Key Takeaway:

- **Best-performing compression regime:**
  8-bit weights, 8-bit activations, 0.02–0.10 sparsity, LR ≈ 0.02
  → **Accuracy = 0.8574**, which is *higher than FP32 baseline* (0.835)
- **Weight_bits = 4 (CR = 8.124×)** is **too aggressive** → accuracy collapses to ~0.80 or worse.
- **Weight_bits = 6 (CR = 5.416×)** gives moderate accuracy (0.806–0.845).
- **Activation bits remained fixed at 8**, and that is essential — **lowering activation bits breaks accuracy badly** in MobileNetV2 QAT without advanced tricks.
- **Overall trend:**
  Compression is well-tolerated until **final_sparsity > 0.3** or **weight_bits < 8**.

Final sparsity	Floor frac	lr	CR weights	Final Qat acc	Model Ram quant_mb	Qat train loss	Qat Val acc	Weights Quant mb
0.1031	0.00825	0.0203	4.062	**0.8574**	18.8037	0.3974	0.8574	2.1005
0.04445	0.00657	0.02169	4.062	0.8564	18.8037	0.40136	0.8564	2.1005
0.02431	0.00124	0.02036	4.062	0.8553	18.8037	0.39859	0.8553	2.1005
0.04527	0.00434	0.02030	4.062	0.8545	18.8037	0.40039	0.8545	2.1005
0.06721	0.00110	0.02015	4.062	0.8527	18.8037	0.40106	0.8527	2.1005
0.02011	0.00554	0.02298	4.062	0.8527	18.8037	0.40056	0.8527	2.1005
0.05201	0.00834	0.02407	4.062	0.8524	18.8037	0.40533	0.8524	2.1005
0.33507	0.00225	0.02105	4.062	0.8521	18.8037	0.40314	0.8521	2.1005
0.18213	0.00949	0.02286	4.062	0.8516	18.8037	0.40390	0.8516	2.1005
0.10496	0.00095	0.02155	4.062	0.8513	18.8037	0.39883	0.8513	2.1005
0.04956	0.00901	0.02522	4.062	0.8512	18.8037	0.40683	0.8512	2.1005
0.11864	0.01598	0.02412	4.062	0.8503	18.8037	0.40765	0.8503	2.1005
0.04109	0.00411	0.02619	4.062	0.8498	18.8037	0.40794	0.8498	2.1005
0.22361	0.00306	0.02225	4.062	0.8498	18.8037	0.40455	0.8498	2.1005
0.67536	0.01980	0.02293	4.062	0.8490	18.8037	0.42320	0.8490	2.1005
0.06051	0.00613	0.04215	4.062	0.8469	18.8037	0.42737	0.8469	2.1005
0.02420	0.01129	0.04955	5.416	0.8454	18.2786	0.44010	0.8454	1.5754
0.28789	0.00892	0.06143	4.062	0.8422	18.8037	0.44941	0.8422	2.1005
0.39993	0.00772	0.03133	8.124	0.8329	17.7535	0.44473	0.8329	1.0503
0.13268	0.00702	0.09906	4.062	0.8257	18.8037	0.50223	0.8257	2.1005
0.45074	0.00235	0.04241	8.124	0.8091	17.7535	0.46318	0.8091	1.0503
0.12000	0.01339	0.14164	5.416	0.8062	18.2786	0.57292	0.8062	1.5754
0.61540	0.01070	0.12152	4.062	0.8054	18.8037	0.55561	0.8054	2.1005
0.54498	0.00777	0.14558	5.416	0.7957	18.2786	0.59306	0.7957	1.5754
0.21958	0.01211	0.16764	8.124	0.7509	17.7535	0.69386	0.7509	1.0503

Question 4. Compression Analysis:

(a) Compression ratio of the model (weights + activations in RAM):
Definition: CRmodel=model_ram_fp32_b / model_ram_quant_b

Weight_bits	Model_ram_fp32_mb	model_ram_quant_mb (computed)	CR_model
8	75.084	18.8037	**3.993×**
6	75.084	18.2786 (1.575375 + 0.0651 + 16.6381)	**4.106×**
4	75.084	17.7538 (1.05025 + 0.0651 + 16.6381)	**4.229×**

Interpretation: Lowering weight bits increases weight compression and slightly improves overall model RAM compression.

(b) Compression ratio of the weights:
Definition: CRweights=weights_fp32_b / weights_quant_b

weight_bits	weights_fp32_mb	weights_quant_mb (computed)	CR_weights
8	8.5323	2.1005	**4.062×**
6	8.5323	1.5754	**5.416×**
4	8.5323	1.0503	**8.124×**

Interpretation: weights_quant_mb was computed by scaling the 8-bit quant MB linearly with bits/8 when the run did not report quant MB directly.

(c) Compression ratio of the activations:
Definition: CR acts=acts_fp32_b / acts_quant_b
How activations were measured

- An ActivationLogger was attached to the model that registers forward hooks on convolutional and linear modules.
- For a single representative batch from the test loader, the logger:
    - Sums numel * 4 bytes for FP32 activations to get acts_fp32_b.
    - Sums ceil(numel * act_bits / 8) bytes for quantized activations and adds a small per-tensor metadata (4 bytes) to get acts_quant_b + meta.
- The values reported in sweep are:
    - acts_fp32_mb = 66.5518 MB
    - acts_quant_meta_mb = 16.6381 MB (this includes quantized activation bytes + small per-tensor meta)
- **Compression ratio (activations)** for all three weight bit settings (activations kept at 8 bits) is:
    - CRacts=66.5518/16.6381≈4.000x

Notes

- Activation compression depends on act_bits and the batch used for measurement.
  I used a single representative batch (same batch size as evaluation) to estimate peak activation memory for inference.
  For different batch sizes, scale activation bytes proportionally.

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

Variable	Definition
weights_fp32_mb	Total size of all model weight tensors stored in 32-bit floating point
weights_quant_mb	Size of all quantized weight bits (before metadata)
weight_meta_mb	Metadata for weight quantization (scales, per-channel floats, small headers)
acts_fp32_mb	Activation memory in FP32 for a representative forward pass (single batch)
acts_quant_meta_mb	Activation quantized bytes plus small per-tensor metadata for the same forward pass
model_ram_fp32_mb	In-memory RAM footprint in FP32 for weights + activations (single batch)
model_ram_quant_mb	In-memory RAM footprint after quantization (weights quant + weight meta + activation quant+meta)
compressed_on_disk_mb	Approximate on-disk size of the compressed artifact (.qmod) containing quantized weights and scales
cr_weights	Compression ratio for weights
cr_acts	Compression ratio for activations
cr_model	Overall model RAM compression ratio

We provide two useful size notions:

1. **On-disk compressed artifact size (approx)**
    - Computed as: weights_quant_mb + weight_meta_mb (excludes tiny headers).
2. **In-memory RAM footprint for inference (weights + activations)**
   what matters for runtime memory usage.
    - Computed as: model_ram_quant_mb = weights_quant_mb + weight_meta_mb + acts_quant_meta_mb.

Computed values

weight_bits	Weights quant_mb	Weight meta_mb	Compressed on_disk_mb	Model ram quant_mb (in memory)
8	2.1005	0.0651	2.1656 MB	18.8037 MB
6	1.5754	0.0651	1.6405 MB	18.2786 MB
4	1.0503	0.0651	1.1154 MB	17.7538 MB

Interpretation

- **On-disk size**: dropping from ~2.17 MB (8-bit) to ~1.12 MB (4-bit) for the quantized weights + scales
  a meaningful storage saving.
- **In-memory size**: dominated by activation memory (≈16.64 MB quantized) so reducing weight bits yields only modest additional RAM savings unless activation bits are also reduced or batch size is lowered.

Compression Summary:

Weight bits	weights FP32 (MB)	weights quant (MB)	weight meta (MB)	On disk compressed (MB)	acts FP32 (MB)	acts quant+ meta (MB)	model RAM FP32 (MB)	model RAM quant (MB)	CR_weights	CR_acts
8	8.5323	2.1005	0.0651	2.1656	66.5518	16.6381	75.084	18.8037	4.062×	4.0×
6	8.5323	1.5754	0.0651	1.6405	66.5518	16.6381	75.084	18.2786	5.416×	4.0×
4	8.5323	1.0503	0.0651	1.1154	66.5518	16.6381	75.084	17.7538	8.124×	4.0×

Question 5. Reproducibility & Repository:

(a) Repo details:

repo/
- train.py # trains baseline MobileNetV2, saves fp32 checkpoint, optional wandb sweep creation
- compress.py # loads fp32 checkpoint, runs pruning+QAT pipeline, exports .qmod + state_dict
- test.py # single-run reproducer (already done)
- utils.py # shared helpers: dataloaders, model builder, quant/prune helpers
- artifacts/ # saved checkpoints, qmod files
- requirements.txt
- README.md

(b) README File:

```
Commands:

pip install -r requirements.txt (include torch, torchvision, pyyaml, numpy, wandb etc.,)

Make sure wandb login is done before running the script to log data.

Baseline training:
   # saves: ./artifacts/baseline_fp32_best.pth
   python train.py --config config.yaml

Compression run:
   # saves final compressed .qmod and .state_dict.pth in ./artifacts
   1. Modify configuration in the config.yaml file
   2. python compress.py --config config.yaml --fp32 ./artifacts/baseline_fp32_best.pth
   3. To enable wandb logging set use_wandb: true in config.yaml or pass --use_wandb to compress.py.
The code will attempt to log artifacts but will not fail if wandb is not configured.
   4. Other model paramters can also be changes in config.yaml file itself.

Test run:
   # or use --qmod to load compressed artifact if you added that option
   1. Copy the compressed model from ./artifacts/mobilenetv2_compressed_...state_dict.pth if
compress.py was run and paste in current    directory as mobilenetv2_compressed_state_dict.pth
   2. python test.py --state mobilenetv2_compressed_state_dict.pth

NOTE: Sample run artifacts are placed inside the artifacts folder. The actual model files are in the current
directory itself.
   1. best_mobilenetv2_epoch99_val83.84.pth -> BASELINE MODEL
   2. mobilenetv2_compressed_model.qmod -> CUSTOM COMPRESSED REPRESENTATION OF QUANTIZED
WEIGHTS
   3. mobilenetv2_compressed_state_dict.pth -> MODEL PARAMETERS IN FP32 MODEL
```

(c)Git Hub Repo:

REPO LINK **(https://github.com/cs24m530/Mobilenet-v2-Assignment3/tree/master)**