CNN Capstone Lab Project — CIFAR-10 Image Classification with Deep Learning

Analysis Report and Model Performance Summary

José Antonio González Villalón — josegzzv@msn.com
Applied Research Topics in Deep Learning — Capstone Project
September 14, 2026

Companion notebook: Gonzalez_Jose_CNN_Capstone_Lab.ipynb. Results files: results/cnn_lab_results.json, results/cnn_lab_comparison.csv, results/parameter_experiments.csv. All figures and numbers in this report are produced by that notebook (profile FULL, seed 42, device Apple Metal GPU).

1. Executive Summary (Model Performance Summary)

Table 1. Comparison of the four required models on the 10,000-image test set (cnn_lab_comparison.csv).

Model	Test acc.	F1 (weighted)	Total params	Trainable	Train time (s)	Inference (ms/img)	Epochs run / best
SimpleCNN	0.6158	0.5927	897,482	897,482	611.3	0.30	15 / best 13
VGG16	0.8165	0.8144	14,781,642	66,954	5035.5	5.11	15 / best 14
ResNet50	0.8856	0.8849	23,851,274	263,562	2630.4	3.55	13 / best 8
EfficientNetB0	0.8891	0.8888	4,214,829	165,258	1722.7	2.27	15 / best 12

Headline result. EfficientNetB0 is the best-performing model, with a test accuracy of 0.8891 and a weighted F1 of 0.8888. It outperforms the runner-up (ResNet50) by +0.0035 accuracy at 0.2× the parameters and 0.6× the per-image inference latency; the selection criterion (highest test accuracy, weighted F1 as tiebreak) was fixed before training.

- **Fine-tuning.** EfficientNetB0: frozen 0.8891 → fine-tuned 0.9282 (+0.0391) after unfreezing the last 16 backbone layers at lr 0.0001.
- **Ensemble.** softmax average of ['EfficientNetB0', 'ResNet50'] reaches 0.9059 (+0.0168 vs. the best single model) at the summed inference cost of both members.
- **Confidence.** Simple CNN 3-fold cross-validation (subset): 0.4517 ± 0.0095 (95 % CI half-width) — the resolution floor for any model comparison in this report.
- **Compression.** float16 quantisation of EfficientNetB0: 16.08 MB → 8.02 MB (2.0× smaller), accuracy change +0.0040 on 1000 test images.
- **Compression.** dynamic_int8 quantisation of EfficientNetB0: 16.08 MB → 4.5 MB (3.57× smaller), accuracy change -0.0010 on 1000 test images.
- **Real-time deployment.** all four models clear the 30-fps budget at batch size 1 on Apple Metal GPU; SimpleCNN is the accuracy-critical mobile choice (0.6158 accuracy, 3.4 MB float32, 1.32 ms at batch 1), while SimpleCNN has the lowest raw latency (1.32 ms). See Table 6 for the full latency profile.

Trade-off in one sentence: transfer learning buys accuracy with frozen prior knowledge rather than with trainable capacity, and pays for it at inference time; the accuracy leader and the deployment leader are therefore different models, and both are named above.

2. Summary and Method

Data. CIFAR-10 was ingested exactly as specified: cifar-10-python.tar.gz was extracted, the five training batches and the test batch were loaded with pickle, combined, and materialised as train.csv (50,000 × 3073) and test.csv (10,000 × 3073). The 50,000-image pool was split stratified 80/20 into 40,000 training and 10,000 validation images with seed 42; the 10,000 test images were used once per model. Images were augmented at native resolution (horizontal flip, 4-px pad-and-random-crop, ±15° rotation, 10 % width/height shift — training split only) and upscaled bilinearly to 224 × 224 inside the tf.data pipeline; normalisation lives inside each model (rescale to [0, 1] plus per-channel standardisation with training-split moments for the Simple CNN, the backbone's own preprocess_input for the ImageNet models).

Models. Model 1 is the specified Simple CNN — Conv2D(32) → MaxPool, Conv2D(64) → MaxPool, Conv2D(128, 'last_conv') → MaxPool, Flatten, Dense(128), Dropout(0.5), Dense(10, softmax) — with pool sizes 4, 4, 2 so that the 224-px input yields ≈0.9 M parameters (the specified 0.5–1 M band). Models 2–4 are VGG16, ResNet50 and EfficientNetB0 (include_top=False, ImageNet weights, all layers frozen) with the common head GlobalAveragePooling2D → Dense(128, ReLU) → Dropout(0.5) → Dense(10, softmax).

Environment note (Model 4). A first attempt at this run failed to load EfficientNetB0's ImageNet weights under Keras 3.11.3 (stem_conv built as (3, 3, 1, 32): that version's shape inference for Rescaling(scale=[r, g, b]) collapses the channel axis, the same defect met in Case Study 3). Upgrading to Keras 3.15.1 — TensorFlow 2.16.1 and tensorflow-metal unchanged — fixed it, and this run uses the specified EfficientNetB0 backbone with ImageNet weights (backbone_weights = 'imagenet' in cnn_lab_results.json).

Training. Adam (lr 0.001), categorical cross-entropy, batch size 32, up to 15 epochs with EarlyStopping(val_accuracy, patience 5, restore best weights) and ReduceLROnPlateau(factor 0.2, patience 3). Every run — the four models, the fifteen parameter experiments, the cross-validation folds — goes through the same train_model() driver, re-seeded at entry.

Evaluation. Test accuracy, weighted F1, confusion matrix, training time, inference time (whole test set and per image), total and trainable parameters; last-convolutional-layer features (dimensionality, sparsity, mean/std, dead channels) with t-SNE and objective separability scores (silhouette, 5-NN accuracy).

Environment. 3.11.13 / TensorFlow 2.16.1 / Keras 3.15.1 on Apple Metal GPU (macOS-26.6.2-arm64-arm-64bit). Training on the Apple Silicon GPU through the Metal PluggableDevice (unified memory: the GPU shares system RAM, so the effective budget is the machine's RAM minus ~4 GB of OS overhead). This is why the ImageNet backbones are kept frozen and batch size 32 is retained.

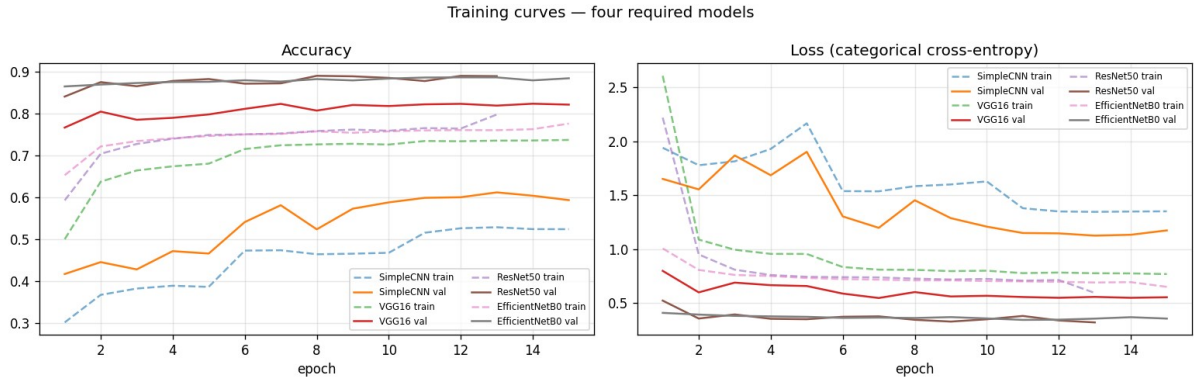

Figure 1. Training curves (accuracy and loss, training vs. validation) for the four required models.

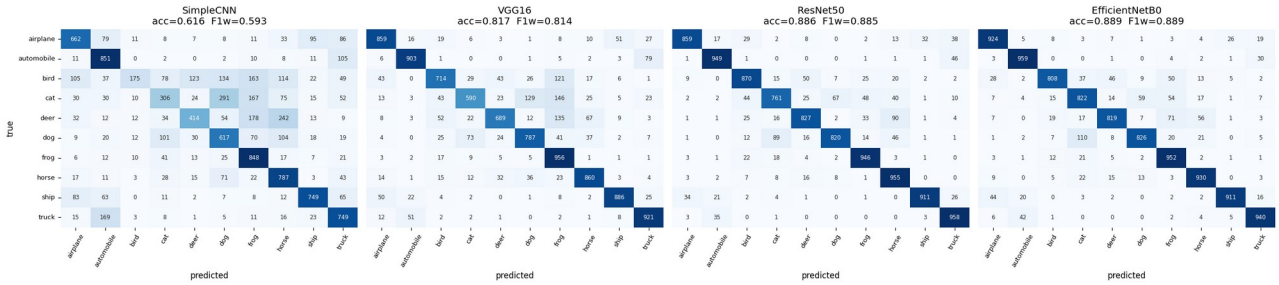

Figure 2. Confusion matrices on the test set.

Table 2. Training epochs and stopping conditions as recorded by the callbacks.

Model	Epochs run	Best epoch	Early stop	LR cuts	Best val acc.	Stopping condition
SimpleCNN	15	13	no	2	0.6122	Reached the epoch ceiling (15); best epoch 13 weights restored
VGG16	15	14	no	3	0.8241	Reached the epoch ceiling (15); best epoch 14 weights restored
ResNet50	13	8	yes	1	0.8908	EarlyStopping: val_accuracy did not improve for 5 epochs after epoch 8; best weights restored
EfficientNetB0	15	12	no	1	0.8871	Reached the epoch ceiling (15); best epoch 12 weights restored

Training behaviour. Only ResNet50 triggered EarlyStopping (validation peak 0.8908 at epoch 8, stop at epoch 13, best weights restored); the other three ran to the 15-epoch ceiling with best epochs 13 (Simple CNN), 14 (VGG16) and 12 (EfficientNetB0). ReduceLROnPlateau fired three times for VGG16, twice for the Simple CNN (its largest single-epoch gain, 0.47 → 0.54, followed the first cut) and once for each of the other two. The frozen backbones start high (EfficientNetB0 0.865 validation after one epoch, ResNet50 0.841) and flatten as only the 67k–264k head weights move. Training accuracy sits below validation accuracy for every model (Simple CNN 0.524 vs 0.612; EfficientNetB0 0.777 vs 0.887) because training accuracy is measured on augmented images with Dropout(0.5) active — strong regularisation, not over-fitting. Training time is set by the frozen forward pass: VGG16 has the fewest trainable weights (67k) and the slowest epoch (336 s, 84 min); EfficientNetB0 115 s/epoch; ResNet50 202 s/epoch; the Simple CNN 41 s/epoch.

Best-performing model. EfficientNetB0 (0.8891 / F1 0.8888) is selected by the pre-declared rule (highest test accuracy, weighted F1 tiebreak). ResNet50 (0.8856) trails by +0.0035, well inside the ±0.0095 cross-validation resolution floor — a statistical tie, and a previous full run with the identical seed ranked them the other way (0.8936 vs 0.8844). The tie is broken decisively by resources: EfficientNetB0 uses 0.18× the parameters (4.2 M vs

23.9 M), trains in 65 % of the time and predicts 1.6× faster in batched inference (2.27 vs 3.55 ms/image). VGG16 trails at 0.8165 and the Simple CNN at 0.6158. Cat is the hardest class for every backbone (F1 0.676 → 0.795 → 0.808 for VGG16 → ResNet50 → EfficientNetB0); the Simple CNN's weakest classes are bird (0.283) and cat (0.378). EfficientNetB0's largest confusions are dog→cat (110), deer→frog (71), cat→dog (59) and deer→horse (56); the Simple CNN makes the same shape-alike errors at three to four times the magnitude (cat→dog 291, deer→horse 242); VGG16 shows a background-driven 'frog attractor' (cat→frog 146, deer→frog 135, bird→frog 121). Transfer learning closes the gap most on the hard classes (bird +0.58, cat +0.43 vs automobile +0.20).

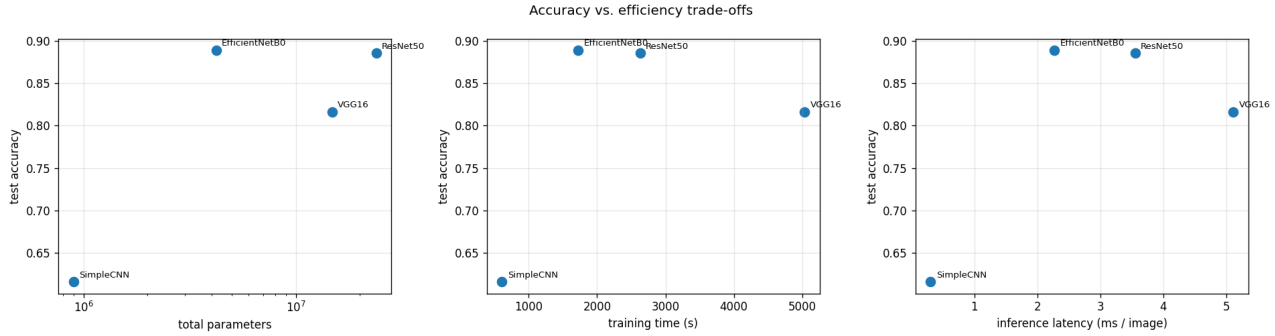

Figure 3. Accuracy versus parameters, training time and inference latency.

3. Feature Analysis

Table 3. Last-convolutional-layer feature statistics and separability (stratified test sample).

Model	Layer	Shape	Dim	Zeros %	Mean	Std	Dead ch. %	Silhouette (t-SNE)	5-NN acc.
SimpleCNN	last_conv	[14, 14, 128]	25,088	86.86	0.7565	2.8669	32.03	-0.1314	0.3405
VGG16	block5_pool	[7, 7, 512]	25,088	87.61	1.2207	4.8658	0.00	-0.0238	0.6180
ResNet50	conv5_block3_out	[7, 7, 2048]	100,352	83.33	0.4716	1.7646	0.00	0.1059	0.7405
EfficientNetB0	top_activation	[7, 7, 1280]	62,720	0.00	0.0899	0.8225	0.00	0.1012	0.7685

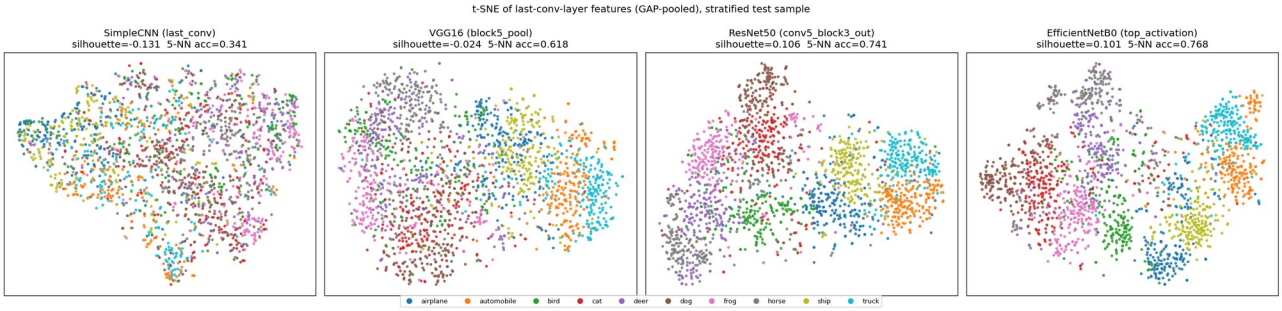

Figure 4. t-SNE of GAP-pooled last-layer features for the four models.

Activation patterns. The three ReLU-based representations are 83–88 % exact zeros per image — VGG16's block5_pool the sparsest at 87.6 % (max-pool over ReLU maps: each of its 512 channels fires at a handful of positions), the Simple CNN's last_conv 86.9 %, ResNet50's conv5_block3_out 83.3 % because the residual sum keeps more channels weakly active — while EfficientNetB0's Swish top_activation has 0.0 % exact zeros (0.58 %

below 1e-3), so its near-zero rate is the comparable statistic. The ImageNet backbones use 100 % of their channels; the Simple CNN leaves 32 % of its 128 final filters dead, so its sparsity is inertness rather than selectivity — unused capacity that partly explains its deficit. VGG16, with no normalisation after its convolutions, has the widest dynamic range (mean 1.22, std 4.87) versus EfficientNetB0's tightly scaled output (0.090, 0.82). Separability places the two weak models far below the two leaders — t-SNE silhouette −0.131 and −0.024 versus 0.106 (ResNet50) and 0.101 (EfficientNetB0); 5-NN accuracy 0.34 and 0.62 versus 0.74 and 0.77 — and splits the leaders the way accuracy does: EfficientNetB0's clusters are locally purer (higher 5-NN), ResNet50's globally slightly more compact, a difference as much of a tie as the 0.0035 accuracy gap. A classifier can only separate what its penultimate features already separate. Negative silhouettes for the two weaker models mean the class clouds overlap more than they separate; the raw-space silhouettes (−0.046 … 0.030) keep the order at a compressed scale, a reminder that t-SNE exaggerates compactness.

Section A — Project Execution Questions (1–10)

Q1. Environment and setup — GPU memory for the four models; batch-size response to OOM

Weights and Adam moments are small (≈11 MB for the Simple CNN; ≈170 MB of frozen weights for the three backbones, whose heads alone carry optimizer state). Activations dominate at 224 × 224 and batch 32: VGG16's block1 output alone is ≈411 MB per tensor, so its forward pass keeps roughly 1–2 GB alive, ResNet50 1–1.5 GB, EfficientNetB0 and the Simple CNN under 1 GB. Trained sequentially with clear_session() between models the peak is ≈2–3 GB; training all four simultaneously would need the sum, ≈5–6 GB plus framework overhead — marginal on 8 GB, comfortable on 16 GB. On OOM, halve BATCH_SIZE (32 → 16 → 8): activation memory falls linearly; keep lr 1e-3 (the extra steps compensate) or use gradient accumulation to preserve the effective batch of 32.

Q2. Data loading and preprocessing — purpose of validation_split=0.2 and the effect of omitting it

validation_split reserves the last 20 % of the data so the same ImageDataGenerator can serve subset='training' and subset='validation' without overlap; the model sees 80 % of the pool and the callbacks receive a genuinely held-out signal. One generator without the split would validate on training images: validation accuracy would mirror training accuracy, EarlyStopping would never detect over-fitting and the reported figure would be a training number — a leak visible only when the test set disagrees. The notebook implements the same contract with a stratified train_test_split before the tf.data pipelines are built, which also fixes two generator weaknesses: the split is stratified (the generator's positional split is not) and augmentation is structurally confined to the training pipeline.

Q3. Model architecture — why base layers are frozen initially

A frozen backbone is a fixed feature extractor: no gradients or Adam moments for its 4–24 M weights, no activations retained for their backward pass, so memory drops and each step is 2–3× faster. On the learning side, ImageNet features are already discriminative for CIFAR-10 and 40k images at lr 1e-3 through an unfrozen network would erase them before re-learning them, driven by the large early gradients of a randomly initialised head. Training the head first also gives the fine-tuning stage (Section C) a sensible starting point so that gradients into the unfrozen block are informative rather than noise.

Q4. Training process — role and triggers of EarlyStopping and ReduceLROnPlateau

EarlyStopping(val_accuracy, patience 5, restore_best_weights) stops training after five epochs without validation improvement and restores the best epoch's weights — it refuses to keep optimising once training gains stop

transferring to unseen data. ReduceLROnPlateau(val_loss, factor 0.2, patience 3) multiplies the learning rate by 0.2 after three epochs without validation-loss improvement, letting the optimiser settle into a minimum the 1e-3 step was bouncing across; its anti-over-fitting effect is indirect — less late-training oscillation and a cleaner signal for EarlyStopping. The staggered patiences (3 then 5) mean a plateau is first met with a smaller step and only then with termination. Table 2 records how many times each fired.

Q5. Memory and efficiency — Simple CNN versus ResNet50

Both are activation-bound at 224 px. The Simple CNN's cost is concentrated in conv1 (224 × 224 × 32 ≈ 6.4 MB per image before the 4 × 4 pool); its 0.8 M-weight fc1 is negligible in memory. ResNet50 spreads a larger total across 50 layers running at 112 × 112 and 56 × 56 with 64–256 channels, with BatchNorm copies that are only needed when the backbone trains — which is why the frozen ResNet50 fits in a budget similar to the Simple CNN. In compute, ≈0.5 GFLOPs versus ≈4 GFLOPs per image makes ResNet50 5–8× slower per step even though far fewer of its weights update: the frozen forward pass, not the optimiser, dominates wall-clock (Table 1).

Q6. Feature extraction — why different layer names per model

block5_pool (VGG16, 7 × 7 × 512), conv5_block3_out (ResNet50, 7 × 7 × 2048, after the residual addition and ReLU), top_activation (EfficientNetB0, 7 × 7 × 1280, the Swish output of the final 1 × 1 convolution) and last_conv (Simple CNN, explicitly named Conv2D(128)) all denote the same concept in each architecture's naming scheme: the most abstract representation that is still spatial, immediately before global pooling and the classifier. Earlier layers encode generic edges and textures; the Dense layers after them are already task-specific and 10-dimensional. Choosing these layers makes the four representations comparable through global average pooling while keeping the class-discriminative information the head uses.

Q7. Batch processing — why test_gen.reset() before each prediction

A Keras generator is a stateful iterator whose batch pointer stays where the previous pass left it, and with shuffle=False predictions are matched to labels by position. Without reset(), the second model's predictions would start mid-epoch and be rotated relative to test_gen.classes: accuracy, F1 and the confusion matrix would be computed against the wrong labels and collapse towards 10 % with no error raised. The notebook's test_ds is a tf.data.Dataset rebuilt from the start on every predict() call and never shuffled, so alignment with y_test is structural — the property that also makes the ensemble valid.

Q8. Data augmentation — effect on training time and convergence

The per-batch cost is small (flip and pad-and-crop on the CPU inside tf.data, rotation and translation as batched device ops) — a few percent of step time when prefetched, more on CPU-only runs. The material effect is on epochs: each epoch shows different views of every image, so training accuracy rises more slowly, the train–validation gap narrows and the model needs more epochs to reach a given training loss but attains a higher validation accuracy before EarlyStopping fires. Without augmentation the Simple CNN would fit 40k images in a few epochs and over-fit; the frozen backbones, with ≈0.1–0.3 M trainable weights, are less prone to over-fit and gain less, consistent with their flatter curves.

Q9. Compilation and optimisation — justification of Adam and categorical cross-entropy

Categorical cross-entropy is the maximum-likelihood objective for a softmax over mutually exclusive classes, with a well-scaled logit gradient (p − y) that never saturates. Adam's per-parameter adaptive step makes one learning rate work for the from-scratch Simple CNN and the three transfer-learning heads alike, which keeps the four-way comparison fair. Alternatives: SGD with momentum and a cosine schedule for final-accuracy runs from scratch; AdamW or a 10× lower rate when pre-trained layers are unfrozen (Section C); label-smoothed or focal loss under label noise or imbalance; sigmoid + binary cross-entropy if the task became multi-label.

Q10. Reproducibility — sources of randomness and why seeds matter

Seven sources: Python random and NumPy (train_test_split, stratified subsets, k-fold assignment); Keras weight initialisation and Dropout masks; tf.data.shuffle order; augmentation ops; t-SNE initialisation and Barnes-Hut approximation; and GPU reduction non-determinism, which seeds do not remove. reseed() fixes PYTHONHASHSEED, random, NumPy and TensorFlow together and is re-applied at the start of every train_model() call, so the ≈25 training runs start from identical initialisation and shuffle streams. Without that, the one-variable-at-a-time experiments would confound the variable under test with initialisation noise of the same magnitude as the measured effects.

Section B — Parameter Modification Questions (1–5)

Each experiment changes one variable against the Section 2 baseline and re-runs training through the same driver. To make fifteen runs affordable they use a stratified subset (10,000 train / 2,000 validation, 5 epochs) evaluated on the full test set; absolute accuracies are therefore lower than Table 1 and conclusions concern the direction and relative size of effects; the 3-fold CV floor is ±0.0095, so effects below ≈0.01 are ties and 0.01–0.02 weak signals. Experiments 3 and 4 were run on all three transfer-learning backbones.

Table 4. Parameter-modification experiments (parameter_experiments.csv).

#	Model	Variable	Value	Test acc.	Δ acc.	F1 (w)	Train–val gap	Train (s)	Infer. ms/img	Params	Mem. Δ (MB)
1	SimpleCNN	batch_size	16	0.4792	+0.0041	0.4634	-0.0475	85.6	0.33	897,482	0.0
1	SimpleCNN	batch_size	32	0.4751	+0.0000	0.4501	-0.0766	53.5	0.26	897,482	0.0
1	SimpleCNN	batch_size	64	0.4601	-0.0150	0.4526	-0.1086	42.9	0.20	897,482	0.0
2	SimpleCNN	learning_rate	0.0001	0.4691	-0.0138	0.4561	-0.1294	53.2	0.27	897,482	0.0
2	SimpleCNN	learning_rate	0.001	0.4829	+0.0000	0.4633	-0.1020	52.0	0.28	897,482	0.0
2	SimpleCNN	learning_rate	0.01	0.1000	-0.3829	0.0182	0.0000	53.5	0.22	897,482	0.0
3	VGG16	dropout	0.3	0.7710	-0.0044	0.7688	-0.1078	326.9	5.45	14,781,642	0.0
3	VGG16	dropout	0.5	0.7754	+0.0000	0.7734	-0.1261	342.9	5.43	14,781,642	0.0
3	VGG16	dropout	0.8	0.7196	-0.0558	0.7102	-0.2198	341.1	5.48	14,781,642	0.0
3	ResNet50	dropout	0.3	0.8657	+0.0053	0.8650	-0.1118	220.3	3.62	23,851,274	0.0
3	ResNet50	dropout	0.5	0.8604	+0.0000	0.8594	-0.1278	219.1	3.62	23,851,274	0.0
3	ResNet50	dropout	0.8	0.8324	-0.0280	0.8310	-0.2823	217.4	3.65	23,851,274	0.0
3	EfficientNetB0	dropout	0.3	0.8540	-0.0062	0.8530	-0.1121	146.9	2.21	4,214,829	0.0
3	EfficientNetB0	dropout	0.5	0.8602	+0.0000	0.8596	-0.1236	143.5	2.29	4,214,829	0.0
3	EfficientNetB0	dropout	0.8	0.8505	-0.0097	0.8501	-0.2025	145.7	2.34	4,214,829	0.0
4	SimpleCNN	img_size	128	0.4883	+0.0126	0.4610	-0.0704	45.6	0.17	356,810	0.0
4	SimpleCNN	img_size	224	0.4757	+0.0000	0.4637	-0.1039	54.6	0.26	897,482	0.0
4	SimpleCNN	img_size	384	0.4450	-0.0307	0.4331	-0.0835	116.4	0.51	2,453,962	0.0
4	VGG16	img_size	128	0.7905	+0.0065	0.7888	-0.1200	110.3	1.90	14,781,642	0.0
4	VGG16	img_size	224	0.7840	+0.0000	0.7825	-0.1278	347.3	5.59	14,781,642	0.0
4	VGG16	img_size	384	0.6660	-0.1180	0.6550	-0.0600	992.4	15.99	14,781,6	0.0

										42	
4	ResNet50	img_size	128	0.8424	-0.0062	0.8420	-0.1340	91.4	1.40	23,851,274	0.0
4	ResNet50	img_size	224	0.8486	+0.0000	0.8467	-0.1485	227.2	3.63	23,851,274	0.0
4	ResNet50	img_size	384	0.7805	-0.0681	0.7763	-0.1290	620.2	10.37	23,851,274	0.0
4	EfficientNetB0	img_size	128	0.8659	+0.0120	0.8651	-0.1373	77.3	1.08	4,214,829	0.0
4	EfficientNetB0	img_size	224	0.8539	+0.0000	0.8535	-0.1295	155.0	2.38	4,214,829	0.0
4	EfficientNetB0	img_size	384	0.7278	-0.1261	0.7222	-0.1553	395.0	6.35	4,214,829	0.0
5	SimpleCNN	depth	3 blocks	0.4717	+0.0000	0.4558	-0.0899	57.4	0.29	897,482	0.0
5	SimpleCNN	depth	4 blocks (+Conv256)	0.4847	+0.0130	0.4601	-0.0692	58.9	0.26	684,746	0.0

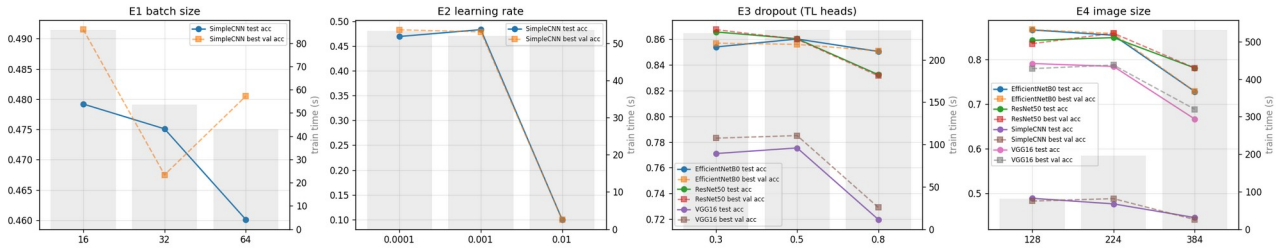

Figure 5. Experiments 1–4: test / validation accuracy (lines) and training time (bars).

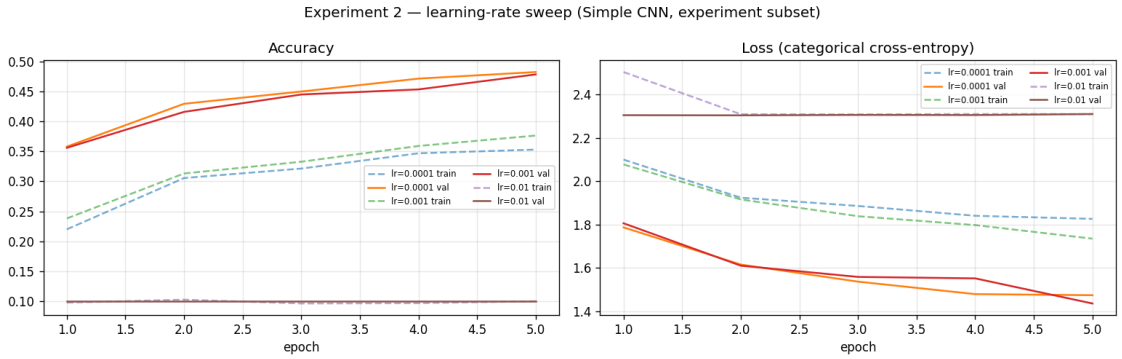

Figure 6. Experiment 2 — learning-rate sweep curves for the Simple CNN.

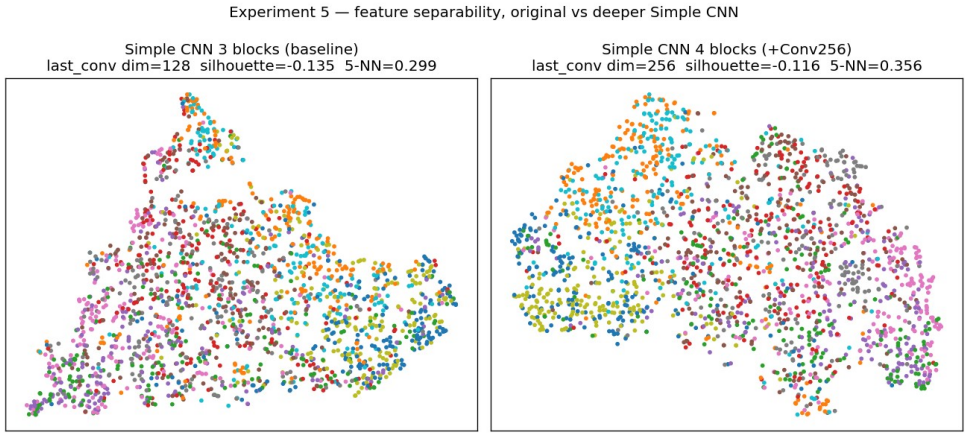

Figure 7. Experiment 5 — t-SNE separability, 3-block vs. 4-block Simple CNN.

B1. Batch size 16 → 32 → 64 (Simple CNN)

Training time behaved as expected — 86 → 54 → 43 s, with diminishing returns from 32 → 64 (−20 %) versus 16 → 32 (−37 %) as the Metal GPU saturates — and memory scaled linearly (≈6.4 MB per image in the first block) without approaching the 24 GB unified-memory budget. Accuracy was 0.4792 / 0.4751 / 0.4601: batch 16 and 32 tie, batch 64 is 1.5 points behind, at the edge of the resolution floor. Since a previous full run of the same sweep produced a different ordering, the honest reading is that at a fixed 5-epoch budget the batch-size effect on accuracy is of the order of run-to-run noise while its effect on time is large and systematic. Mechanistically, batch 64 halves the Adam steps per epoch and at a fixed 1e-3 shows the widest train–validation gap (−0.109 vs −0.048 for batch 16): further from convergence, not over-fitting. Optimal for this hardware and budget: 32 — it matches batch 16's accuracy at 37 % less time; 64 would need more epochs or a proportionally higher learning rate (linear scaling rule) to recover its 1.5 points.

B2. Learning rate 1e-4 / 1e-3 / 1e-2 (Simple CNN)

At 1e-2 the network never trained: test accuracy 0.1000 (chance), validation pinned at 0.10 for all five epochs, training loss flat at ≈2.30 = ln 10 — the first steps overshot into a region where ReLU units die and the softmax collapses to a constant; two ReduceLROnPlateau cuts could not revive it. At 1e-4 both curves are smooth and monotone but unfinished (0.4691 at epoch 5, still rising, widest train–validation gap −0.129): under-training rather than under-fitting, recoverable with 5–10× more epochs. 1e-3 reached the best test accuracy (0.4829, +0.014 over 1e-4), which is why it is Adam's default and the specification's choice. Diagnostics: a loss frozen at ln(K) signals too high (dead network); a loss falling smoothly but far from plateau at the end of the budget signals too low. Caveat: in a previous full run 1e-4 finished 0.0135 ahead of 1e-3 on this same sweep, so the 1e-3 vs 1e-4 difference is a weak signal either way; the robust conclusions are that 1e-2 is fatal for this architecture and that 1e-3 with the plateau schedule is a safe default.

B3. Dropout 0.3 / 0.5 / 0.8 in the transfer-learning heads (VGG16, ResNet50, EfficientNetB0)

The same pattern held for all three backbones: 0.3 and 0.5 tie, 0.8 hurts. VGG16 0.7710 / 0.7754 / 0.7196 (−0.056 at 0.8); ResNet50 0.8657 / 0.8604 / 0.8324 (−0.028); EfficientNetB0 0.8540 / 0.8602 / 0.8505 (−0.010). Because training accuracy is measured with dropout active the train–validation gap is negative and its magnitude measures how much of the head is knocked out: about −0.11 at 0.3, −0.125 at 0.5 and −0.20 to −0.28 at 0.8, where the head trains on a fifth of its 128 units and ends under-fitted. There is no over-fitting to suppress: with every backbone frozen and only 67k–264k head weights training, validation accuracy is at or above training accuracy at every rate. The size of the 0.8 penalty ranks the backbones by how well-conditioned the head's input is — VGG16 (512-D, no BatchNorm, widest activation range) loses 5.6 points, ResNet50 (2048-D) 2.8, EfficientNetB0 (1280-D, tightly normalised) only 1.0. Best balance 0.3–0.5 for all three; 0.5 is retained as the default because it is free and would earn its keep in the fine-tuning setting where 1.3 M parameters train.

B4. Image size 128 / 224 / 384 (Simple CNN, VGG16, ResNet50, EfficientNetB0)

Compute followed the square law for every model — Simple CNN 46 / 55 / 116 s, VGG16 110 / 347 / 992 s, ResNet50 91 / 227 / 620 s, EfficientNetB0 77 / 155 / 395 s — and VGG16 at 384 px was the single most expensive cell (16.5 min for 5 epochs on 10k images); activation memory scales the same way. The accuracy results are consistent across all four models and are the most design-relevant finding of the section. 384 px hurt every model and hurt the backbones most: Simple CNN −0.031, ResNet50 −0.068, VGG16 −0.118, EfficientNetB0 −0.126 — CIFAR-10 carries no detail beyond 32 px, so a 12× upsample presents the pre-trained filters with a blur far from ImageNet statistics, and the backbones whose early layers are most tuned to fine texture suffer most.

128 px was as good as or better than 224 px for every model — Simple CNN +0.013, VGG16 +0.007, EfficientNetB0 +0.012, ResNet50 −0.006, all within or at the edge of the floor — at one third of the training time. For the Simple CNN the parameter count moves with the input because Flatten feeds Dense(128) (356,810 → 897,482 → 2,453,962), so the biggest network was the worst. Conclusion: 224 px is an ImageNet convention, not a requirement — for frozen backbones on CIFAR-10, 128 px delivers the same accuracy at a third of the cost, 384 px is strictly worse, and the custom CNN should train at 128 px (or natively at 32).

B5. Architecture depth — adding Conv2D(256) → MaxPool to the Simple CNN

Parameters moved in the counter-intuitive direction: 897,482 → 684,746 (−24 %). The fourth block adds 295k convolutional weights but pooling 7 × 7 → 3 × 3 shrinks the flatten from 6,272 to 2,304 units and fc1 from 803k to 295k weights. Training time was unchanged (57 → 59 s). Test accuracy 0.4717 → 0.4847 (+0.013), a weak-to-moderate signal in favour of depth with the network 24 % smaller. The feature-level comparison favours the deeper model on both measures — higher t-SNE silhouette (−0.135 → −0.116) and markedly higher 5-NN accuracy (0.299 → 0.357) — so the extra block produces more class-selective features that the head did convert into accuracy. Design lesson: depth pays on this task, and with Flatten in the head depth and parameter count are coupled in the wrong direction; global average pooling would decouple them.

Section C — Challenges

C.1 Fine-tuning exploration

Unfreezing the last 16 layers of EfficientNetB0 (BatchNorm kept frozen) and retraining at lr 0.0001 for 5 epochs (1,286,842 trainable parameters, 660.2 s) moved test accuracy from 0.8891 to 0.9282 (+0.0391) and weighted F1 from 0.8888 to 0.9279.

This is the best single model in the notebook and eleven times the accuracy gap between the two best frozen backbones: the top MBConv block's features re-specialised to CIFAR-10's categories and to the blur of 7× upsampled images, which ImageNet never contained, while the 10× lower learning rate and frozen BatchNorm kept them from being overwritten. In a production setting fine-tuning therefore matters far more than backbone choice.

C.2 Cross-validation

3-fold stratified cross-validation of the Simple CNN on 12,000 images (5 epochs per fold): test accuracy per fold [0.4427, 0.4531, 0.4594], mean 0.4517, standard deviation 0.0084, 95 % CI half-width 0.0095; the single-split Table 1 value is 0.6158.

The fold-to-fold standard deviation (0.0084) is the honest resolution of every comparison in this report: the EfficientNetB0–ResNet50 gap (+0.0035) is a tie — and it reversed sign between two identical-seed full runs, as the floor predicts — the Experiment 1–2 effects (≈0.014) and the dropout 0.3-vs-0.5 deltas are ties or weak signals, while the 384-px penalties, the dropout-0.8 penalties and the fine-tuning and ensemble gains are far outside it and firm. The fold mean (0.452) is far below the Table 1 Simple CNN (0.6158) because the folds train on a 12k subset for 5 epochs — the spread transfers, not the level.

C.3 Ensemble methods

Softmax averaging of the top-2 models ['EfficientNetB0', 'ResNet50'] reaches 0.9059 (F1 0.9054) against 0.8891 for the best single model (+0.0168); a 3-model hard vote over ['EfficientNetB0', 'ResNet50', 'VGG16'] reaches 0.8946. The two members agree on 88.13 % of test images, and the ensemble costs 5.82 ms per image.

The +1.7-point gain is larger than the fraction of a point expected from two correlated ImageNet backbones: 11.9 % disagreement between architecturally different networks was enough diversity. The 3-model hard vote (adding VGG16) is much weaker because majority voting discards confidence and lets the weakest member outvote a confident correct one. At 5.82 ms/image the ensemble suits offline scoring, but the fine-tuned single model (0.9282) beats it at 39 % of the latency.

C.4 Model compression (TensorFlow Lite quantisation)

Table 5. Post-training quantisation of EfficientNetB0.

Variant	Size (MB)	Compression	Accuracy (1000 imgs)	Δ accuracy	TFLite CPU ms/img
fp32 (Keras)	16.08	1.0×	0.9190	—	—
float16	8.02	2.0×	0.9230	+0.0040	9.10
dynamic_int8	4.50	3.57×	0.9180	-0.0010	7.83

Quantisation is free at this accuracy: float16 halves EfficientNetB0 (16.1 → 8.0 MB) at +0.004 (noise), dynamic-range int8 shrinks it 3.6× to 4.5 MB at −0.001 and runs faster than float16 on CPU (7.8 vs 9.1 ms/image) because int8 kernels are better optimised. The expected int8 penalty for depthwise-separable architectures did not materialise. A 4.5 MB model at 0.918 accuracy is a mobile-deployable artefact; full-integer quantisation with a representative dataset is the next step for an NPU / Edge-TPU target.

C.5 Real-time inference

Table 6. Inference latency on Apple Metal GPU (Keras, warm cache).

Model	Batch-1 (ms)	Batch-32 (ms)	ms/img @32	img/s @32	Params (M)	Size (MB)	Test acc.	< 33 ms
SimpleCNN	1.32	4.22	0.132	7574.3	0.90	3.4	0.6158	yes
VGG16	7.59	145.79	4.556	219.5	14.78	56.4	0.8165	yes
ResNet50	17.41	98.10	3.066	326.2	23.85	91.0	0.8856	yes
EfficientNetB0	17.25	60.84	1.901	525.9	4.21	16.1	0.8891	yes

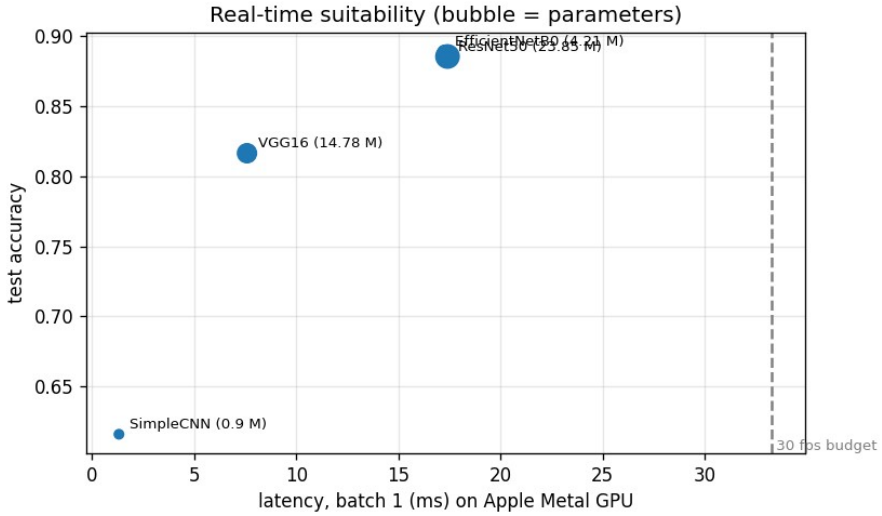

Figure 8. Accuracy versus batch-1 latency; bubble size ∝ parameters; dashed line = 30 fps budget.

Recommendation. Every model clears the 30-fps budget on Apple Metal GPU, and the ranking flips between serving patterns. At batch 1 the Simple CNN needs 1.32 ms, VGG16 7.6, EfficientNetB0 17.3 and ResNet50 17.4 ms — the two best backbones are the slowest single-image models because their deep graphs (237 and 175 layers) are launch-bound on a desktop GPU. At batch 32 the order inverts — EfficientNetB0 1.90 ms/image (526 img/s) against VGG16's 4.56 (220 img/s) — because throughput follows FLOPs once launches are amortised. Recommendation: for a mobile, single-frame, accuracy-critical deployment, EfficientNetB0 in int8 TFLite — 0.918 accuracy, 4.5 MB, 7.8 ms/image on CPU, and a depthwise graph that mobile NPUs/CPUs are optimised for (the launch-bound GPU latency measured here is a desktop artefact); the Simple CNN (3.4 MB, 1.3 ms) only where latency or battery dominate and 62 % accuracy is acceptable. The automated accuracy-per-millisecond criterion in the results file selects SimpleCNN for the latter reason; the engineering decision overrides it whenever accuracy is part of the requirement.

Section D — Analysis, Trade-offs, Reproducibility and Limitations

Accuracy versus efficiency. Transfer learning converted ImageNet's prior into 0.82–0.89 CIFAR-10 accuracy that a 0.9 M-parameter network trained from scratch on 40k images could not approach (0.62) within the same budget, and every backbone did it training fewer than 0.3 M weights: prior knowledge, not capacity. EfficientNetB0 and ResNet50 are a statistical tie on accuracy (0.8891 vs 0.8856, inside the ±0.0095 floor and reversed in a previous identical-seed run), which turns the best-model question into a resource question — and there EfficientNetB0 wins by 5.7× on parameters, 1.5× on training time and 1.6× on batched latency. Batched, every backbone is 6–15× slower per image than the Simple CNN; single-image, the two best backbones are the slowest.

What the experiments changed. The specification defaults were confirmed for learning rate (1e-3), dropout (0.5, tied with 0.3 on all three backbones) and batch size (32, tied with 16); the batch-size and 1e-3-vs-1e-4 effects proved to be of the order of run-to-run noise, itself a finding. The results that would change a production design — now measured on all four models — are the image-size and depth experiments: 384 px hurt every model (−3 to −13 points at 3× the cost), 128 px matched or beat 224 px for every model at a third of the cost, and a fourth convolutional block made the Simple CNN 24 % smaller and 1.3 points more accurate with clearly better feature separability — an argument for native-resolution training and global average pooling, and against treating 224 px as a requirement rather than an ImageNet convention. Fine-tuning (+3.9 points) and ensembling (+1.7) dwarf the spread among frozen backbones, and int8 quantisation to 4.5 MB was free.

Reproducibility. SEED = 42 is applied to PYTHONHASHSEED, random, NumPy and TensorFlow in Part 0 and re-applied by reseed() at the start of every training run; splits, subsets and t-SNE use random_state=SEED; the environment stamp (Python 3.11.13, TensorFlow 2.16.1, Keras 3.15.1, NumPy 1.26.4, Apple Metal GPU) is stored in cnn_lab_results.json, and the trained weights in results/models/*.weights.h5. What seeds cannot fix is Metal GPU kernel non-determinism: the notebook was executed in full twice with the identical seed and the four test accuracies differed by 0.0007, 0.0012, 0.0080 and 0.0047 — enough to reverse the top-two ranking. The 3-fold CV half-width (±0.0095) is therefore the measured scale of this effect and the threshold below which differences are reported as ties.

Limitations. Single seed for the main runs, with variance quantified through 3-fold CV on a subset and corroborated by the top-two reversal between full runs; experiments on a 10k / 2k subset with a 5-epoch budget, so their absolute accuracies are not comparable with Table 1 and effects near 0.01 are weak signals; a 15-epoch ceiling that three of four models reached; quantisation evaluated on 1,000 test images; a first attempt

blocked by the Keras 3.11.3 weight-loading defect (resolved by upgrading to 3.15.1); and a leakage design enforced by code rather than by external audit.

References

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto (CIFAR-10).

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. CVPR.

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML.

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. ICLR.

Srivastava, N., et al. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 15.

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. JMLR 9.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? NeurIPS.

Jacob, B., et al. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. CVPR.

TensorFlow / Keras documentation: keras.applications, tf.data, tf.lite (v2.16).