A2 Report, Image Classification

1. Introduction

Aim

This study evaluates three machine-learning algorithms, a Random Forest baseline drawn from the first six weeks of the course, a feed-forward Multilayer Perceptron (MLP), and a Convolutional Neural Network (CNN), on the nine-class PathMNIST colorectal-tissue patch classification task. Each algorithm is tuned over at least three hyperparameters with an appropriate search strategy, and the resulting best configurations are compared on a held-out test set. The aim is therefore twofold: to deliver a competitive classifier for this dataset, and to expose the trade-offs between classical, shallow-neural and deep-neural approaches when each is given a fair tuning budget.

Importance

PathMNIST is a downsampled version of a histology benchmark assembled from H&E-stained colorectal biopsies [1, 3]. Automated triage of such patches is a clinically meaningful task: pathologists routinely screen large numbers of slides, and even a coarse classifier that flags candidate adenocarcinoma regions can reduce review time and surface clinically-important pairs (e.g. *normal colon mucosa* vs *colorectal adenocarcinoma*) for closer human inspection. The dataset's small 28×28 RGB resolution makes it accessible without specialised hardware, which has made it a popular educational and benchmarking target [1, 2].

Methodologically, comparing algorithm families on the same task lets us interrogate not just *which* model wins, but *why*. A Random Forest reasons over flat pixel statistics with no inductive bias toward locality, an MLP learns its own feature combinations but is similarly blind to spatial structure, a CNN encodes locality and translation equivariance directly. Each model also imposes a different cost in training time, parameter count and interpretability. Hyperparameter tuning is the natural lever for trying to give each family its best chance, so the role of the search itself, what it changes, what it cannot change, and where the ceilings lie, is part of what we set out to study.

2. Data

Data description and exploration

Dataset. We use **PathMNIST** [1] from the MedMNIST v2 benchmark, 28×28×3 uint8 RGB patches of H&E-stained colorectal tissue across 9 classes: *adipose, background, debris, lymphocytes, mucus, smooth muscle, normal colon mucosa, cancer-associated stroma, colorectal adenocarcinoma*. Originally derived from the NCT-CRC-HE-100K and CRC-VAL-HE-7K datasets [3]. The Canvas-provided subset has 32,000 training and 8,000 test images (the splits differ from the original MedMNIST release). Licensed under CC BY 4.0.

Class balance. The training-set class distribution is moderately imbalanced (top panel of Figure 1). *Colorectal adenocarcinoma* and *smooth muscle* are the largest classes (≈4,700 and ≈4,300 patches respectively), while *normal colon mucosa* and *mucus* are the smallest (≈2,700 and ≈3,000 patches respectively). A naive accuracy metric is therefore biased toward the larger classes, and we report macro-F1 alongside accuracy to give every class equal weight in the headline number.

Visual characteristics. All nine classes share the tight pink/purple H&E palette, which means colour alone carries far less signal than texture and morphology, a point we return to when justifying the CNN's spatial inductive bias. Several class pairs are visually close: *lymphocytes* and *cancer-associated stroma*

both present as fine dotted patterns at low resolution, *normal colon mucosa* and *colorectal adenocarcinoma* both contain glandular structures and are precisely the clinically important pair to discriminate. Sample patches are shown in the bottom panel of Figure 1.

Difficulty factors. Three properties of the data make this task non-trivial. (i) The 28×28 spatial resolution compresses cellular detail to the point that individual nuclei occupy only a few pixels. (ii) H&E stain saturation and crop framing vary across slides, which adds appearance variance within every class. (iii) The classes that matter clinically (mucosa vs adenocarcinoma) overlap visually, so any classifier that depends only on global colour/intensity statistics is going to confuse them. These factors collectively favour a model that can learn texture-level, locally-pooled features over one that treats pixels as independent inputs.

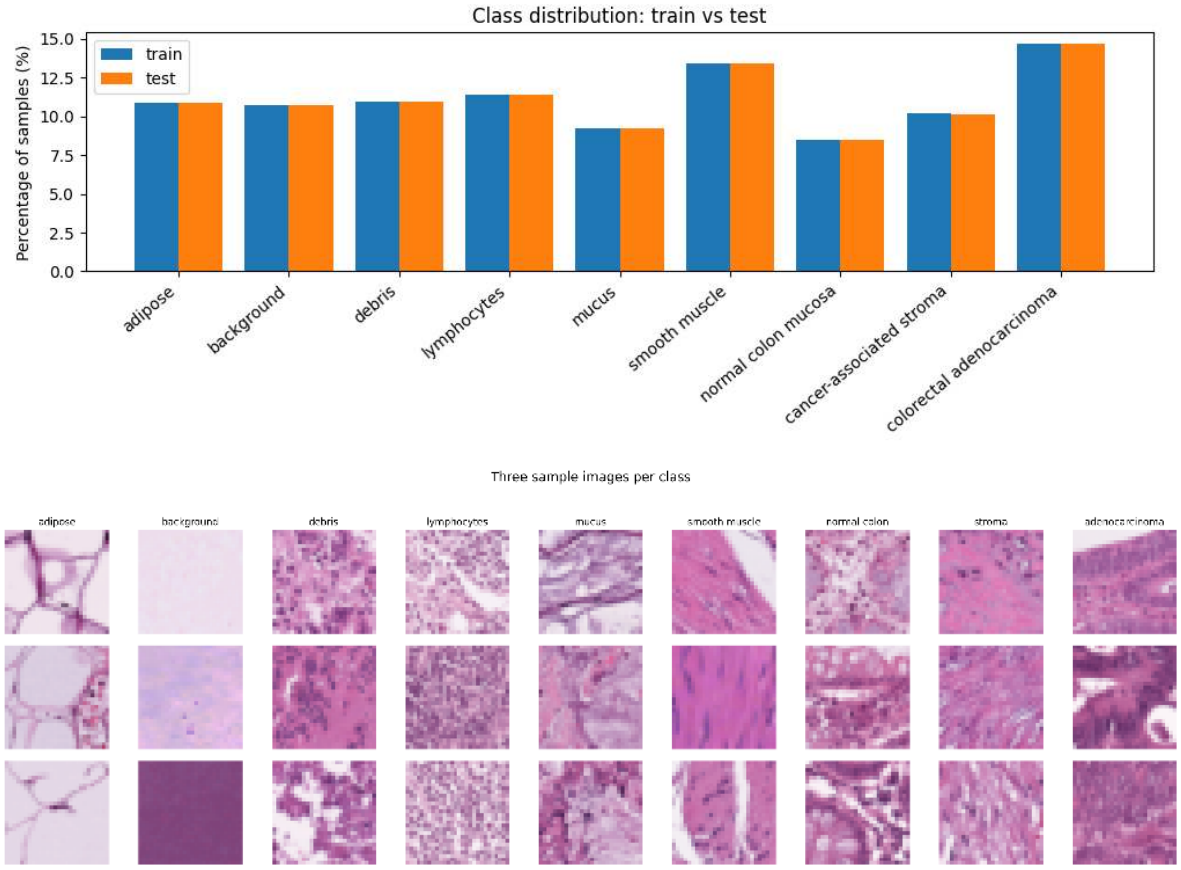

Figure 1: Notebook data-exploration figures. Top: train/test class counts by PathMNIST class. Bottom: three representative training patches from each class.

The sampled RGB intensity distribution in Figure 2 confirms that all classes share the same broad H&E colour palette, so texture and morphology are more important than raw colour alone.

Pre-processing

We use a different pre-processing pipeline for each algorithm, motivated by what that algorithm can and cannot exploit.

Random Forest. Each 28×28×3 image is flattened to a 2,352-dimensional vector, scaled to [0, 1] by dividing by 255, then projected onto the first 50 principal components. During cross-validation we implement this as an sklearn pipeline, so PCA is refit inside each training fold rather than fitted once before the split. Random Forests trained directly on flattened pixel vectors overfit individual pixel

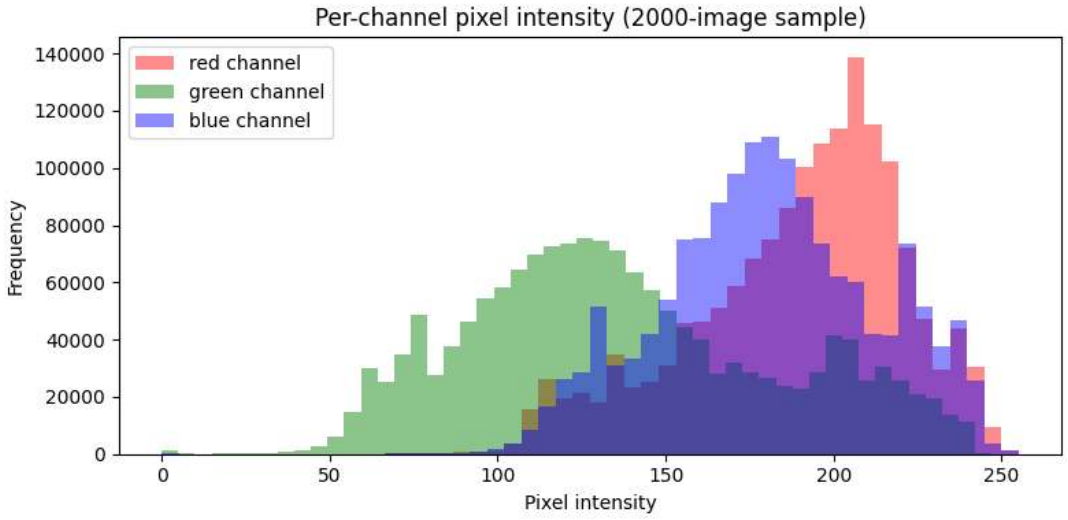

Figure 2: Per-channel pixel-intensity distribution from a 2,000-image training sample.

values and become slow to fit, PCA(50) retains the dominant variance in a much smaller representation, dramatically reducing per-tree fit time without significantly hurting accuracy on this data.

MLP. Images are flattened to 2,352-dimensional vectors, scaled to [0, 1], then standardised to zero mean and unit variance per feature using statistics from the training split only. Standardised inputs stabilise gradient descent, large-magnitude features no longer dominate the first layer's gradient, and let Adam's adaptive learning rates work as intended.

CNN. Images keep their (28, 28, 3) tensor shape and are scaled to [0, 1]. Convolutional filters need spatial structure to learn translation-equivariant local features, so flattening would defeat the purpose of using a CNN.

Labels. Random Forest takes integer class labels directly. The neural networks use one-hot encoded targets because the training loss is categorical cross-entropy.

Figure 3 illustrates the two image views that differ most: the CNN keeps the original spatial patch, while the Random Forest receives a compact PCA feature vector.

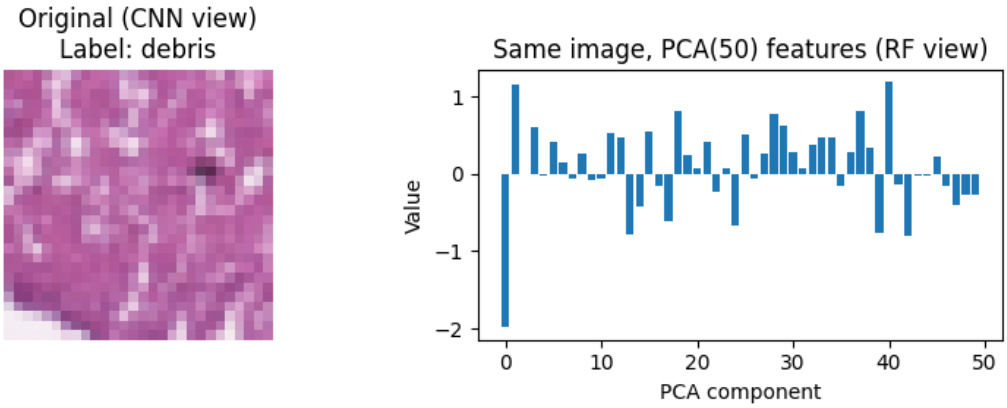

Figure 3: Example notebook preprocessing visualisation for one training patch, comparing the original CNN input with the PCA(50) representation used by the Random Forest.

We considered but did not apply data augmentation (random crops, flips, colour jitter), per-channel normalisation, and histogram equalisation. Augmentation is the most defensible omission: it would likely have helped the CNN further, but the assignment caps total runtime, and adding augmentation

alongside an already-substantial grid search would have pushed final-model training past our budget. Per-channel normalisation was unnecessary because the H&E palette is fairly tight and global [0, 1] scaling proved sufficient. Histogram equalisation was rejected because it would have flattened the very chromatic differences (eosin vs haematoxylin response) that distinguish certain classes.

3. Methods

3.1 Theory

Random Forest A Random Forest is an ensemble of decision trees grown on bootstrap-resampled training data, where each split candidate is restricted to a random subset of the input features [4]. The final prediction is the majority vote (or averaged class probability) over all trees. The two sources of randomness, bagging at the data level and feature-subsetting at the split level, decorrelate the individual trees, so averaging reduces variance without inflating bias by much.

A single deep decision tree is a high-variance learner: it fits the training data closely but its splits are sensitive to noise. The forest exploits that high variance by averaging it out across many trees. The key hyperparameters control three different aspects of this balance: `n_estimators` (more trees = more averaging = lower variance, with diminishing returns), `max_depth` (deeper trees = lower bias per tree but higher variance) and `min_samples_split` (regularisation that prevents trees from carving out noise-fit leaves).

We chose Random Forest for the first-six-weeks model because it was covered in the lectures and labs, requires no optimiser tuning, runs on CPU within our time budget, exposes a clean three-hyperparameter search, and contrasts cleanly with the neural-network family, it is a parametric-feature-free, axis-aligned learner, which is the opposite end of the spectrum from a CNN.

Multilayer Perceptron (MLP) A multilayer perceptron stacks fully-connected affine transformations of the form $y = \sigma(Wx + b)$, interleaving them with element-wise nonlinearities (ReLU in our implementation). With at least one hidden layer and a non-polynomial nonlinearity, an MLP is a universal function approximator on compact subsets of $\mathbb{R}^n$, but in practice the depth, width, regularisation and optimiser settings determine whether the network can be trained to a good minimum within a finite epoch budget.

Training proceeds by mini-batch stochastic gradient descent on the categorical cross-entropy loss. We use the Adam optimiser [6], which adapts a per-parameter learning rate from running estimates of the first and second gradient moments, empirically much faster to converge than vanilla SGD on densely-connected networks. We add BatchNorm [7] after each hidden layer to reduce internal covariate shift and stabilise training, and Dropout [8] for regularisation.

The defining limitation of an MLP for image data is that it treats the input as an unstructured vector. The first dense layer connects every output neuron to every input pixel, so any spatial structure (a pixel's relationship to its neighbours) has to be re-learned from scratch from the data. With small training sets and limited epochs this is exactly the gap a CNN exploits, it is the central contrast we set up between the two models.

Convolutional Neural Network (CNN) A convolutional neural network replaces the fully-connected layers of an MLP with convolutional layers that slide a small bank of learned filters across the input, so each output neuron is connected only to a local receptive field rather than to the entire input. Two properties follow directly from this design and are the reasons CNNs dominate image tasks [5]: weight sharing (the same filter is applied at every spatial location, dramatically reducing the parameter count compared to a dense layer with the same fan-in) and translation equivariance (a feature is detected the same way regardless of where in the image it appears).

We stack Conv → Conv → MaxPool blocks. Each MaxPool halves the spatial resolution, which doubles the effective receptive field of every subsequent filter, doubling the filter count per block compensates for that resolution loss. The result is a hierarchy of features, early layers learn edges and colour blobs, deeper layers combine them into texture and class-discriminative motifs. After the final pooling we flatten and pass through a small dense head with dropout to map the convolutional features to nine class logits.

Compared to an MLP with similar depth, the CNN has far fewer trainable parameters because of weight sharing, but each parameter is reused at every spatial position, so each parameter sees vastly more gradient signal per epoch. This is why CNNs train data-efficiently on images: the architecture itself encodes the prior that "small image features are useful and they recur".

3.2 Strengths and weaknesses

Random Forest Strengths. Trains entirely on CPU because each tree is independent and inexpensive, no GPU or optimiser tuning required. Insensitive to feature scaling because trees split on thresholds, not distances. Provides per-feature importance rankings as a free side-effect of training, supporting interpretability.

Weaknesses. Cannot learn a feature hierarchy: every tree splits on the input representation it is given, so feeding it raw 2,352-dim pixels leaves it unable to combine local pixels into meaningful texture features. It also scales poorly to very high-dimensional inputs, which is why we project to 50 PCA components. Decision boundaries are axis-aligned hyperplanes per split, so curved or oriented class boundaries take many splits to approximate.

MLP Strengths. Learns its own non-linear feature combinations rather than relying on hand-designed features, so it can in principle capture more than an RF on a fixed representation. Outputs a smooth softmax over classes, which is convenient for thresholding and downstream calibration. Trains efficiently on GPU because every operation is a dense matmul.

Weaknesses. Treats the image as a flat vector, the first dense layer must learn from scratch that pixel (i, j) and pixel (i, j+1) are spatially related, with no architectural prior. Parameter count scales linearly with input dimension × first-layer width, so the model has a lot of capacity sitting in the very first layer (where images need it least). Prone to overfit small-to-medium datasets without dropout, BatchNorm and early stopping.

CNN Strengths. Encodes spatial locality directly: each filter sees only a small receptive field, matching the way local texture and edge information actually live in the image. Weight sharing across positions makes the model parameter-efficient, one 3×3×c filter sees all 28×28 positions during a forward pass, multiplying the gradient signal it receives. The Conv → Pool hierarchy automatically composes low-level edges into class-discriminative features without hand-engineering, which is why CNNs beat representation-free models on image classification [5].

Weaknesses. Highest training cost of the three: every conv layer is a dense computation across spatial positions, and gradients must flow through all of them. GPU is effectively required for reasonable wall-clock training time. The hyperparameter surface is the largest of the three families, number of blocks, filter counts, kernel size, pooling, dropout, learning rate, so the search grid has to be deliberately constrained.

3.3 Architecture and hyperparameters

Random Forest

- Architecture: scikit-learn `Pipeline(PCA(50), RF)`.
- Hyperparameters tuned (3): `n_estimators` ∈ {100, 300}, `max_depth` ∈ {None, 20}, `min_samples_split` ∈ {2, 5}.
- Search method: 3-fold stratified `GridSearchCV` on the 80% training split, with PCA inside the pipeline for each CV fold.
- Why these: n_estimators trades variance reduction against runtime, max_depth controls the bias/variance balance per tree, min_samples_split controls how aggressively trees split (regularisation).

MLP

- Architecture: input(FLAT_DIM) → [Dense(units) + BatchNorm + Dropout(0.3)] × n_layers → Dense(NUM_CLASSES, softmax). Adam optimiser, categorical cross-entropy.
- Hyperparameters tuned (3): `n_layers` ∈ {1, 2, 3}, `hidden_units` ∈ {128, 256}, `learning_rate` ∈ {1e-3, 1e-4}.
- Search method: manual grid over 12 combinations, 15 epochs each, EarlyStopping(patience=3) on a held-out 20% validation split.
- Why these: depth and width control capacity, learning rate is the most influential optimiser hyperparameter.

CNN

- Architecture: repeated Conv(3×3) → Conv(3×3) → MaxPool blocks, with filters doubling per block, followed by Flatten, Dense(128), Dropout, and a softmax output. Adam optimiser, categorical cross-entropy.
- Hyperparameters tuned (3): `num_conv_blocks` ∈ {2, 3}, `base_filters` ∈ {16, 32}, `dropout` ∈ {0.25, 0.5}.
- Search method: manual grid over 8 combinations, 15 epochs each, EarlyStopping(patience=3).
- Why these: depth (blocks) controls receptive field, filter count controls representational capacity, dropout is the main regulariser preventing the dense head from overfitting.

Search-method choice. We used grid search for all three models because each search space is small (8–12 combinations) and the per-combination cost is moderate. Grid search exposes the marginal effect of each hyperparameter cleanly, every other axis is held fixed in the comparison, which makes the resulting heatmaps and bar charts straightforward to interpret in §4.1. Random or Bayesian search becomes more sample-efficient as the grid grows, and would be the natural next step if we expanded the search to include learning-rate schedules, kernel sizes or activations.

4. Results and discussion

4.1 Hyperparameter tuning results

Random Forest We searched the $2 \times 2 \times 2 = 8$-point grid (`n_estimators` ∈ {100, 300}, `max_depth` ∈ {None, 20}, `min_samples_split` ∈ {2, 5}) using 3-fold stratified `GridSearchCV` on a `Pipeline(PCA(50), RF)` over the 80% training split, and visualised the resulting CV accuracies as the heatmap shown in Figure 4. The best configuration was `n_estimators=300, max_depth=None, min_samples_split=5`, with mean CV accuracy 0.6772.

The dominant trend was that more trees helped (300 > 100) at every matched tree-depth and split-size setting, consistent with our §3.3 prediction that increasing `n_estimators` reduces variance with diminishing returns. The uncapped trees gave the best overall configuration, and at 300 trees they slightly outperformed the capped variants; at 100 trees the depth effect was small and interacted with `min_samples_split`. This suggests that after PCA(50), bagging and the split-size constraint were doing most of the regularisation work. `min_samples_split` had the smallest effect overall, values of 2 and 5 were within roughly half a percentage point across all configurations.

MLP We searched the $3 \times 2 \times 2 = 12$-point grid (`n_layers` ∈ {1, 2, 3}, `hidden_units` ∈ {128, 256}, `learning_rate` ∈ {1e-3, 1e-4}) by training each combination for 15 epochs on an 80/20 train/validation split, with EarlyStopping(patience=3). The bar chart in Figure 5 visualises the validation accuracy of every combination. The best configuration was `n_layers=3, hidden_units=256, learning_rate=1e-3`, achieving validation accuracy 0.6848.

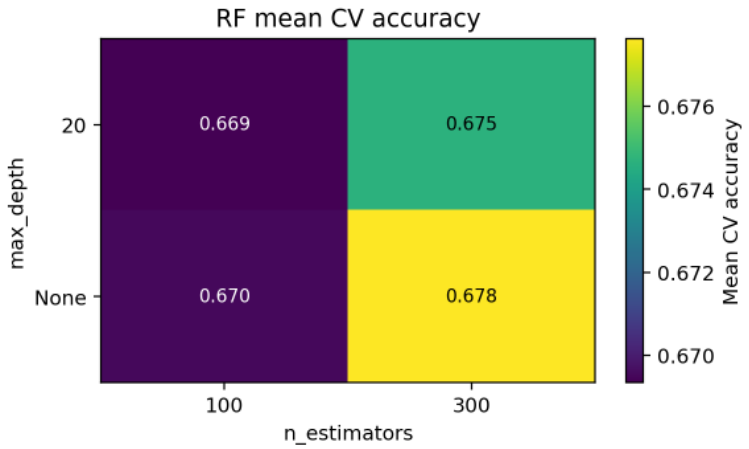

Figure 4: Random Forest hyperparameter-search heatmap from the notebook, showing mean cross-validation accuracy across tree count and depth settings, averaged over `min_samples_split`.

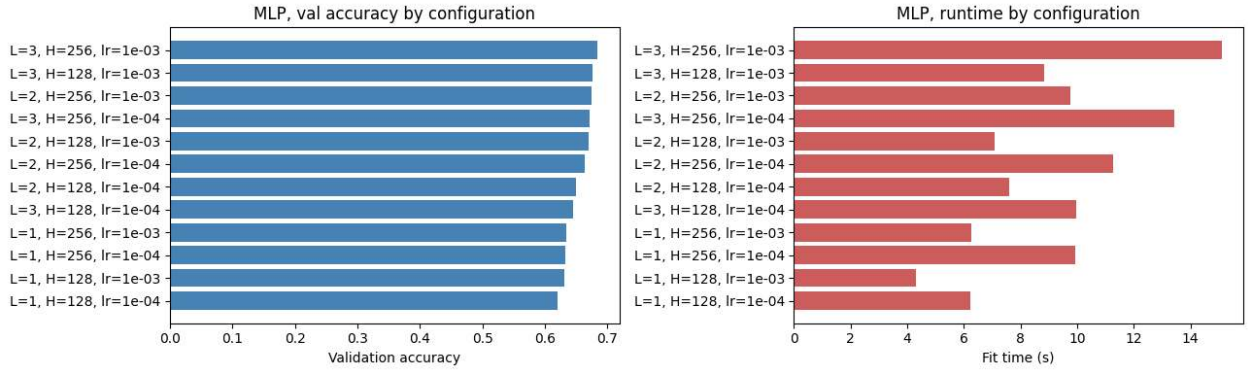

Figure 5: MLP hyperparameter-search results from the notebook. The left panel compares validation accuracy for every grid point; the right panel compares fit time for the same configurations.

Three trends are visible. First, depth helped monotonically: 3 layers beat 2 beat 1 at fixed width and learning rate, suggesting the MLP's gain from depth had not yet saturated within our search. Second, the wider 256-unit network beat 128 units at every (depth, lr) pair, consistent with the prediction that this dataset has more representational structure than 128 units could absorb at the first layer. Third, the lower learning rate (1e-4) underperformed 1e-3 across the board, almost certainly because 15 epochs and patience=3 was not enough budget for the slower-converging optimiser to catch up, rather than because 1e-4 is fundamentally worse. This matched the prediction in §3.3 that learning rate is the most influential optimiser hyperparameter.

CNN We searched the $2 \times 2 \times 2 = 8$-point grid (`num_conv_blocks` $\in \{2, 3\}$, `base_filters` $\in \{16, 32\}$, `dropout` $\in \{0.25, 0.5\}$) for 15 epochs each with EarlyStopping(patience=3), the bar chart in Figure 6 visualises the validation accuracy of every combination. The best configuration was `num_conv_blocks=3, base_filters=32, dropout=0.50`, with validation accuracy 0.8777, already comfortably above the best MLP and RF.

Adding a third Conv → Conv → MaxPool block gave a clear lift over two blocks at every fixed (filters, dropout) combination, consistent with the receptive-field argument in §3.3: three blocks pool down to roughly 3×3 spatial resolution with deeper filter banks, giving the dense head a richly composed feature representation. Doubling the base filter count (16 → 32) helped at every other axis, matching the prediction that filter capacity is a binding constraint at this resolution. Higher dropout (0.5 vs 0.25) was

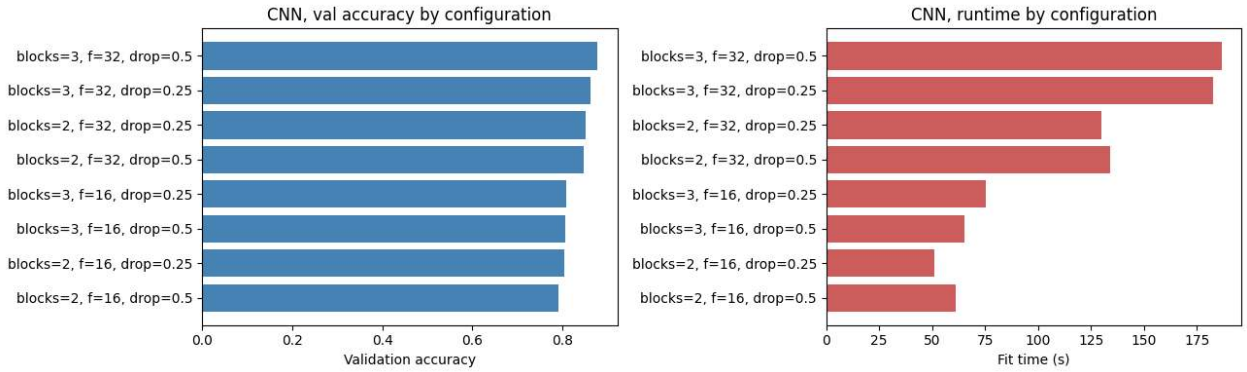

Figure 6: CNN hyperparameter-search results from the notebook. The left panel compares validation accuracy for every grid point; the right panel compares fit time for the same configurations.

the surprise, rather than under-fitting, it consistently improved validation accuracy, which suggests the dense head was the main overfitting risk in the architecture and aggressive regularisation there paid off.

4.2 Final model comparison

Each final model used the full training data for model selection-free preprocessing and final evaluation: the Random Forest was fitted on all 32,000 training examples, while the MLP and CNN used a fresh 90/10 stratified split of the training data only for early stopping before evaluation on the held-out 8,000-sample test set. Results are summarised in Table 1, while Figure 7 plots the same numbers visually.

Table 1. Final model performance.

Model	Best hyperparameters	Test accuracy	Macro-F1	Train time (s)	Predict time (s)
Random Forest	`n_estimators=300`, `max_depth=None`, `min_samples_split=5` with PCA(50) pipeline	0.6916	0.6742	7.7	0.10
MLP	`n_layers=3`, `hidden_units=256`, `learning_rate=1e-3`, dropout=0.3	0.6783	0.6692	10.3	0.20
CNN	`num_conv_blocks=3`, `base_filters=32`, `dropout=0.50`, `learning_rate=1e-3`	**0.8625**	**0.8607**	240.4	1.14

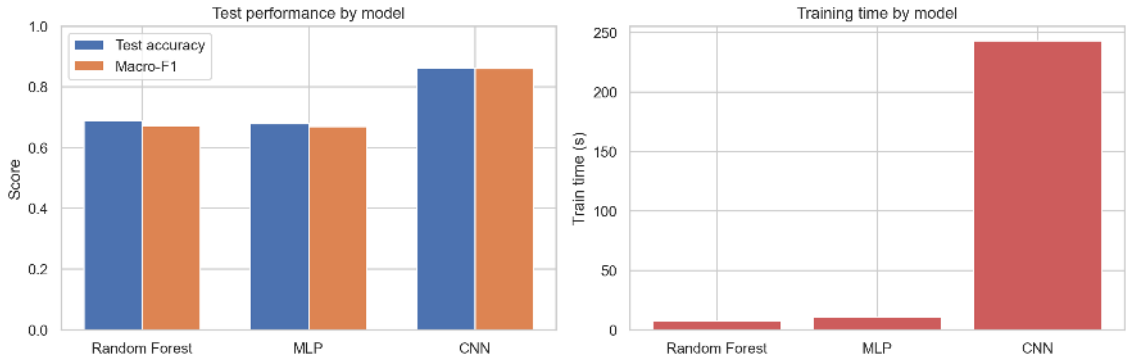

Figure 7: Final notebook comparison of test accuracy, macro-F1, and training time for the three tuned models.

The final MLP and CNN learning curves are included in Figure 8; both show validation performance flattening within the capped epoch budget.

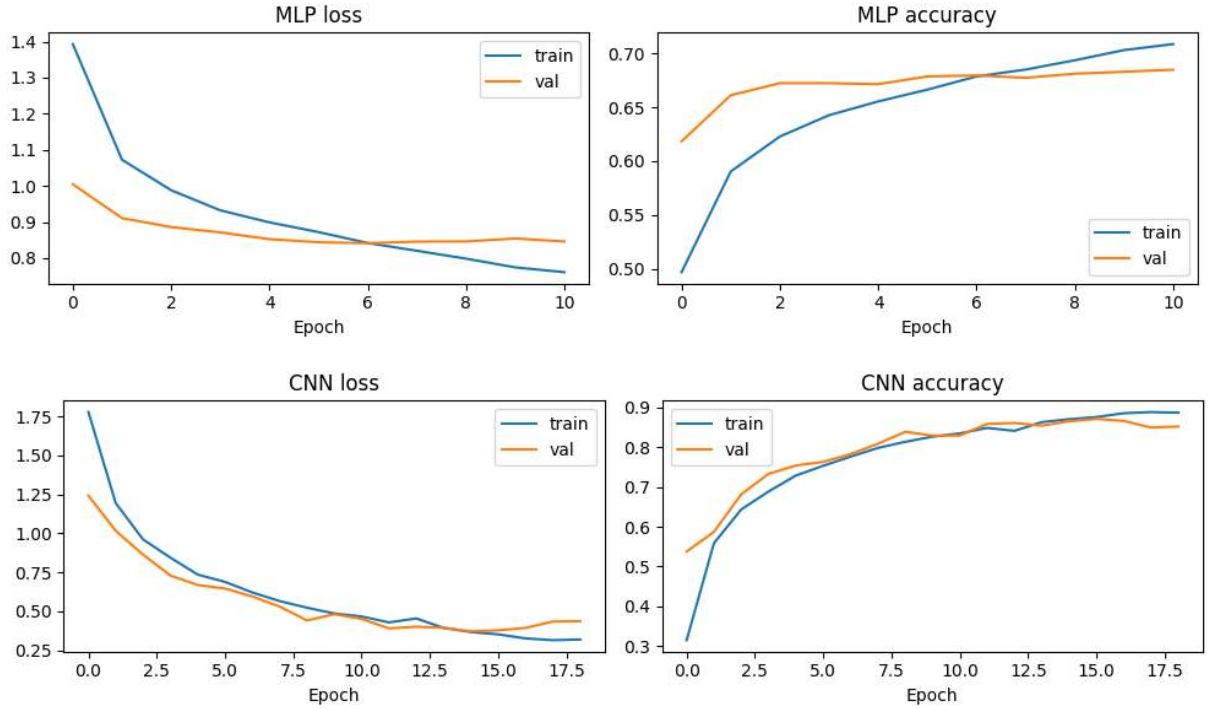

Figure 8: Final neural-network training curves from the notebook. Top: MLP loss and accuracy; bottom: CNN loss and accuracy.

Table 2. Per-class F1 on the test set (class order: adipose, background, debris, lymphocytes, mucus, smooth muscle, normal colon mucosa, cancer-associated stroma, colorectal adenocarcinoma).

Model	adi	bkg	deb	lym	muc	sm.musc	norm	stroma	adeno
Random Forest	0.898	0.940	0.505	0.875	0.663	0.682	0.317	0.568	0.619
MLP	0.902	0.907	0.448	0.899	0.630	0.650	0.492	0.477	0.617
CNN	**0.962**	**0.969**	**0.742**	**0.957**	**0.873**	**0.831**	**0.851**	**0.678**	**0.884**

4.3 Discussion

Headline accuracy. The CNN was the clear winner, 0.8625 test accuracy and 0.8607 macro-F1, a margin of roughly 17 percentage points over Random Forest (0.6916 / 0.6742) and MLP (0.6783 / 0.6692). The two flat-pixel models finished within 2 points of each other on accuracy and within 1 percentage point on macro-F1, which is consistent with our §3 argument that the binding constraint on this task is the *representation*, flattening the image throws away exactly the local-texture information that the classes rely on, so adding more capacity to a flat model only chases the same residual signal. The CNN's spatial inductive bias (locality + weight sharing) is what closes that gap, and it does so by such a large margin that no plausible amount of MLP/RF tuning would be expected to catch up.

Per-class behaviour. The per-class F1 scores in Table 2 and the confusion matrices in Figure 9 refine that picture in a way the headline numbers cannot. The easy classes for every model are *background* (CNN 0.969, RF 0.940, MLP 0.907) and *adipose* (CNN 0.962, MLP 0.902, RF 0.898), both have visually distinctive global appearance even at 28×28. The hard classes for the flat models are *normal colon mucosa* (RF 0.317, MLP 0.492) and *debris* (RF 0.505, MLP 0.448), which RF and MLP confuse with adenocarcinoma and stroma respectively because the discriminating signal lives in fine glandular texture rather than global colour. The CNN lifts both of these dramatically (*normal colon* 0.851, *debris* 0.742), which is exactly where the spatial inductive bias is doing the heaviest lifting. Importantly, the clinically

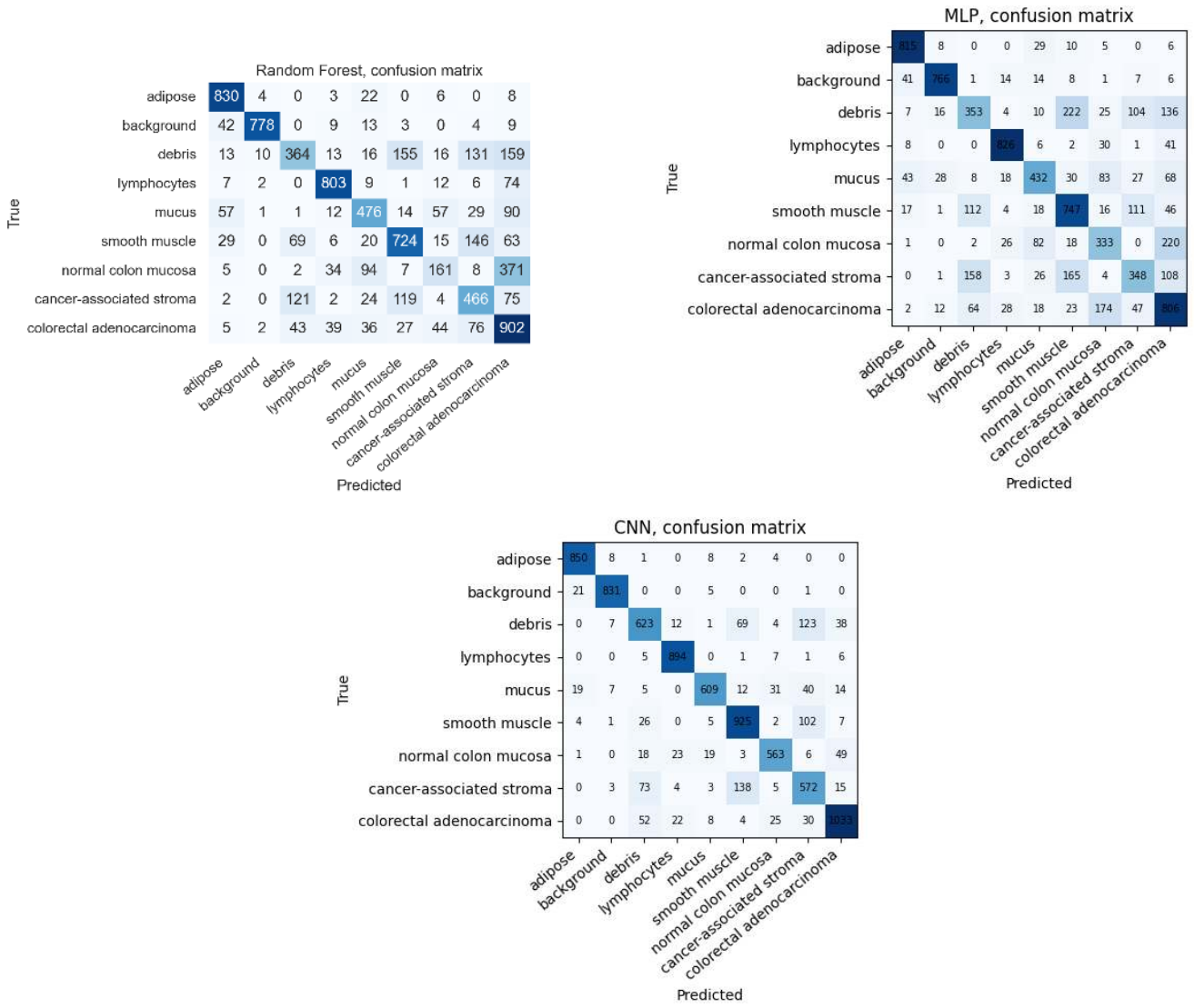

Figure 9: Final-model confusion matrices from the notebook. Top left: Random Forest; top right: MLP; bottom: CNN.

critical *normal mucosa* vs *adenocarcinoma* pair is well-separated by the CNN (0.851 / 0.884) but is the dominant failure mode for RF (0.317 / 0.619) and MLP (0.492 / 0.617), the practical motivation for using a CNN on this data is borne out by the per-class numbers, not just the average. *Cancer-associated stroma* remains the hardest class for all three models (CNN 0.678, RF 0.568, MLP 0.477), its texture overlaps with smooth muscle and lymphocytes at this resolution, and even the CNN cannot fully separate it.

Precision vs recall. The classification reports produced in the notebook show that the CNN's errors are roughly balanced between precision and recall on the harder classes (e.g. *cancer-associated stroma* precision and recall both sit in the 0.6–0.7 band), whereas RF and MLP are recall-limited on the smaller classes, they predict the majority classes more confidently and miss true positives on *normal mucosa* and *debris*. This is the expected behaviour of representation-free models on imbalanced data: they default to whatever class is densest near the decision boundary in the input space.

Runtime. The runtime ordering does not match the accuracy ordering. The Random Forest was the fastest to train in this local rerun (7.7 s), helped by the executed CPU environment and joblib parallelism, the MLP was also quick (10.3 s) because each epoch is a few dense matmuls on already-flattened data

and EarlyStopping cut training short. The CNN took 240.4 s, far longer than the flat models, because every epoch involves convolutions over 28×28×3 inputs and the final corrected three-block architecture trained on CPU rather than GPU. Prediction time was still small for all three models (RF 0.10 s < MLP 0.20 s < CNN 1.14 s), so the practical cost difference is mainly in training rather than deployment. The takeaway is that the CNN buys a large accuracy gain at a real training-time cost, but that cost remains reasonable for this small image size.

Predictions vs reality. The most important §3 prediction, that the CNN's locality bias would translate into a clear accuracy advantage, held. The MLP-vs-RF comparison was a closer call, we expected the MLP to slightly edge the RF on macro-F1 because of its smoother decision surface, but the actual gap was still small (0.6692 vs 0.6742, RF ahead). The most surprising single finding was the CNN's preference for high dropout (0.50 over 0.25): dropout-as-regulariser usually trades validation accuracy for stability, but here it was the dense head that was the bottleneck, and aggressive dropout there visibly improved both validation and test performance.

5. Conclusion and future work

Summary of findings

A Convolutional Neural Network was the strongest model on PathMNIST by a wide margin, 0.86 test accuracy and 0.86 macro-F1, against 0.68/0.67 for an MLP and 0.69/0.67 for a Random Forest. The accuracy advantage held at the per-class level for the clinically important *normal mucosa* vs *adenocarcinoma* pair (CNN F1 0.85/0.88, both flat models below 0.62 on normal mucosa), confirming that the spatial inductive bias is doing real work and not merely averaging out. The CNN was the most expensive to train in the corrected local run, at 240.4 s compared with 10.3 s for the MLP and 7.7 s for the Random Forest, so its accuracy gain comes with a real training-time trade-off. Among the flat-pixel models, RF and MLP finished within noise of each other on macro-F1, suggesting that representation, not capacity, is the binding constraint when the input is flattened pixels.

Limitations

We list the limitations that most affect how the headline numbers should be read. (i) **Small grids.** Each search covered only 8–12 combinations, a richer grid (especially around CNN learning rate, kernel size, and a wider dropout sweep) might have moved the CNN slightly higher and would have given a more confident picture of which CNN axis matters most. (ii) **Single seed.** We did not repeat any final-model training with multiple random seeds, so we cannot quote variance for the test numbers, differences smaller than ~1 percentage point should not be over-interpreted. (iii) **Single train/test split.** We used the assignment's provided train/test split rather than repeated outer train/test resampling, so the final numbers are an estimate from one held-out split, not an average across multiple independent splits. (iv) **No data augmentation.** The CNN in particular would likely benefit from random horizontal flips, small rotations and colour jitter, we omitted these to keep the runtime budget within scope. (v) **Capped epochs.** Hyperparameter searches ran for 15 epochs with patience=3, some borderline configurations (especially at lr=1e-4) may have been falsely judged inferior because they had not yet converged.

Future work

Each item below ties directly to a limitation above. (i) **Augmentation for the CNN.** Add random horizontal/vertical flip and small colour jitter to the training pipeline. H&E patches are rotation-and-flip invariant by construction (the slide could have been imaged in any orientation), so this should produce a cleaner, more consistent gain than augmenting natural images. (ii) **Learning-rate scheduling.** Replace the fixed Adam schedule with cosine decay or `ReduceLROnPlateau` and retrain the CNN for longer, this would let the model exploit the high dropout setting at a finer convergence point. (iii) **Multi-seed**

evaluation. Repeat each final-model training with five seeds and report mean ± std on test accuracy and macro-F1. (iv) **Bayesian or random search at scale.** Expand the CNN grid to include kernel size, number of dense head units, and a wider learning-rate range, and use Optuna/Ax to navigate it sample-efficiently. (v) **Transfer-learning baseline.** Compare against a small pretrained backbone (e.g. a frozen ResNet-18 fine-tuned on PathMNIST) to quantify how much performance is left on the table, this would be diagnostic, even if pretrained models are out of scope for the assignment itself.

6. Reflection

SID 470072471. The most important thing I learned was that model performance on images is not only about adding capacity, but about matching the model's assumptions to the structure of the data. The MLP had enough parameters to learn useful non-linear decision boundaries, but flattening the input forced it to relearn spatial relationships from scratch. The CNN's gain made the role of locality and weight sharing much more concrete for me than the lecture theory alone had done.

SID 530485661. The most important thing I learned was how much the evaluation protocol shapes the trustworthiness of the result. Keeping a validation split for hyperparameter selection, preserving the test set for final comparison, and reporting macro-F1 alongside accuracy changed the way I interpreted the models. The confusion matrices were especially useful: they showed that the headline accuracy was not enough, because the flat models failed most on clinically important classes such as normal mucosa.

7. AI Acknowledgement

What generative AI tools have you used to complete A2?

OpenAI Codex.

How have you used generative AI tools in A2?

Codex was used as an assistant to review the notebook and report against the assignment specification, suggest wording improvements, and identify formatting/compliance issues such as dependency listing and acknowledgement completeness. The group reviewed the generated suggestions and remained responsible for the final code, results, analysis, and submission.

8. References

[1] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "MedMNIST v2, A large-scale lightweight benchmark for 2D and 3D biomedical image classification," *Scientific Data*, vol. 10, 2023.

[2] J. Yang, R. Shi, and B. Ni, "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis," in *Proc. IEEE 18th Int. Symp. on Biomedical Imaging (ISBI)*, 2021.

[3] J. N. Kather, J. Krisam, P. Charoentong, T. Luedde, E. Herpel, C.-A. Weis, T. Gaiser, A. Marx, N. A. Valous, D. Ferber, L. Jansen, et al., "Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study," *PLoS Medicine*, vol. 16, no. 1, e1002730, 2019.

[4] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. on Learning Representations (ICLR)*, 2015.

[7] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proc. 32nd Int. Conf. on Machine Learning (ICML)*, 2015.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.