The tables below were generated programmatically from the instantiated models by 03_Experiments/src/ at revision, not transcribed by hand, so they cannot drift from the code that produced the results. Output shapes are for a batch size of 1.

1. Architectures

TinyCNN (width 32) — PathMNIST primary model (input 3 x 28 x 28)

#	Layer	Configuration	Output shape	Parameters
1	Conv2d	3->32, k=3, s=1, p=1, bias=True	32x28x28	896
2	BatchNorm2d	32 features, eps=1e-05, momentum=0.1	32x28x28	64
3	Conv2d	32->64, k=3, s=1, p=1, bias=True	64x14x14	18,496
4	BatchNorm2d	64 features, eps=1e-05, momentum=0.1	64x14x14	128
5	Conv2d	64->128, k=3, s=1, p=1, bias=True	128x7x7	73,856
6	BatchNorm2d	128 features, eps=1e-05, momentum=0.1	128x7x7	256
7	Dropout	p=0.15	128	0
8	Linear	128->9	9	1,161

Total trainable parameters: 94,857

Lite224CNN — native-resolution and controlled-resolution model (input 3 x 224 x 224)

#	Layer	Configuration	Output shape	Parameters
1	Conv2d	3->16, k=3, s=1, p=1, bias=True	16x224x224	448
2	BatchNorm2d	16 features, eps=1e-05, momentum=0.1	16x224x224	32
3	ReLU	-	16x224x224	0
4	MaxPool2d	k=2, s=2	16x112x112	0
5	Conv2d	16->32, k=3, s=1, p=1, bias=True	32x112x112	4,640
6	BatchNorm2d	32 features, eps=1e-05, momentum=0.1	32x112x112	64
7	ReLU	-	32x112x112	0
8	MaxPool2d	k=2, s=2	32x56x56	0
9	Conv2d	32->64, k=3, s=1, p=1, bias=True	64x56x56	18,496
10	BatchNorm2d	64 features, eps=1e-05, momentum=0.1	64x56x56	128
11	ReLU	-	64x56x56	0
12	MaxPool2d	k=2, s=2	64x28x28	0
13	Conv2d	64->128, k=3, s=1, p=1, bias=True	128x28x28	73,856
14	BatchNorm2d	128 features, eps=1e-05, momentum=0.1	128x28x28	256
15	ReLU	-	128x28x28	0
16	MaxPool2d	k=2, s=2	128x14x14	0

#	Layer	Configuration	Output shape	Parameters
17	Conv2d	128->128, k=3, s=1, p=1, bias=True	128x14x14	147,584
18	BatchNorm2d	128 features, eps=1e-05, momentum=0.1	128x14x14	256
19	ReLU	-	128x14x14	0
20	AdaptiveAvgPool2d	output=1	128x1x1	0
21	Flatten	-	128	0
22	Dropout	p=0.2	128	0
23	Linear	128->9	9	1,161

Total trainable parameters: 246,921

2. Optimisation and training protocol

Setting	TinyCNN	Lite224CNN
Loss function	Cross-entropy (unweighted)	Cross-entropy (unweighted)
Optimiser	AdamW	AdamW
Learning rate	2×10^{-3}	1×10^{-3}
Weight decay	1×10^{-4}	1×10^{-4}
Learning-rate schedule	Cosine annealing, T_max = epochs	Cosine annealing, T_max = epochs
Epochs	15	8
Batch size	512	64
Weight initialisation	PyTorch default	PyTorch default
Numerical precision	float32	float32
Early stopping	None	None
Validation frequency	Once per epoch	Once per epoch
Checkpoint selection	Best validation macro-F1	Best validation macro-F1
Data-loader workers	0	0
Gradient clipping	None	None

Augmentation

Applied to training batches only; validation and test data are never augmented.

Transform	Probability	Applied at
Horizontal flip	0.5	Batch level (TinyCNN), image level (Lite224CNN)
Vertical flip	0.5	Batch level (TinyCNN), image level (Lite224CNN)
Rotation by $k \times 90°$, $k \in \{0,1,2,3\}$	Uniform over k (0.75 that $k \neq 0$)	Batch level (TinyCNN), image level (Lite224CNN)

No colour, brightness, contrast, blur, noise or elastic augmentation was used during training. This matters for interpreting the perturbation results in Table IV(b): the networks were never shown colour-perturbed data, which is one reason their accuracy falls so steeply under those perturbations.

Transparent baselines

Setting	ColorTexture-SGD	ColorTexture-RF
Estimator	SGDClassifier	RandomForestClassifier
Loss / criterion	log_loss	Gini impurity
Preprocessing	StandardScaler	None
Trees / iterations	max_iter = 30, tol = 10^{-3}	260 trees
Class weighting	None	balanced_subsample
Input	91 colour-texture descriptors	91 colour-texture descriptors
Probability path	predict_proba	predict_proba

All models are scored through predict_proba (or softmax for the networks) so that the comparison is not confounded by different decision rules. Using predict instead would change the ColorTexture-SGD accuracy by 0.0006, because scikit-learn's one-vs-all decision_function and the normalised predict_proba disagree on four of 7,180 test patches.

3. Hyperparameter search budget

No hyperparameter search was performed for any model. All settings were fixed a priori from standard defaults before any test-split result was seen, and no model received a larger tuning budget than another. The search space is therefore empty and the reported values are untuned. This is stated in the manuscript Methods and recorded as limitation five, because it bears directly on the fairness of the resource-aware comparison: a tuned TinyCNN and an untuned random forest would not be comparable.

The only architectural variation explored is the width sweep reported in Table II (widths 8, 16, 24, 32, 48), which is presented as a result rather than used to select the primary model. The primary model was fixed at width 32 before the sweep was run; width 24 subsequently proved more accurate, and we report that rather than retrospectively redefining the primary model.

4. Randomness and reproducibility

Aspect	Value
Primary seed	42, fixed before any repeated-seed result was generated
Additional seeds	2024, 7777, 31337, specified at revision before their results were inspected
Earlier seeds	7, 123
Seeded components	random, numpy.random, torch.manual_seed, torch.cuda.manual_seed_all, and the DataLoader shuffling generator
Bitwise determinism	**Not** enforced. torch.use_deterministic_algorithms was not enabled, so convolution and reduction order may vary

Because bitwise determinism is not enforced, the same seed does not reproduce exactly across accelerator backends. Supplementary File S5 quantifies this: across six seeds the mean accuracy agreed to 0.007 between platforms while individual seeds differed by up to 0.021. This is the expected behaviour of non-deterministic GPU kernels and is one of the reasons the manuscript reports multi-seed rather than single-run results.