IFT6135 - Assignment 1 Report

Guillaume Genois, 20248507

February 27, 2026

Problem 2

Question 1: Speed comparison of `discrete_2d_convolution` vs `scipy.signal.convolve2d`

Experimental setup

The average wall-clock time (over 5 runs) is measured for two implementations: a custom function (`discrete_2d_convolution`), implemented in pure NumPy with explicit Python `for`-loops over every output pixel, and the SciPy function (`scipy.signal.convolve2d`). We sweep four image sizes (64, 128, 256, 512 pixels square) and three kernel sizes (3×3, 7×7, 15×15) on random float64 arrays.

Results

Table 1: Mean execution time (seconds) and speedup of SciPy over the custom implementation.

Image size	Kernel	Custom (s)	SciPy (s)	Speedup
64×64	3×3	0.0230	0.0001	$\approx 201\times$
64×64	7×7	0.0215	0.0004	$\approx 58\times$
64×64	15×15	0.0219	0.0013	$\approx 17\times$
128×128	3×3	0.0840	0.0004	$\approx 219\times$
128×128	7×7	0.0843	0.0015	$\approx 56\times$
128×128	15×15	0.0878	0.0053	$\approx 17\times$
256×256	3×3	0.3294	0.0018	$\approx 186\times$
256×256	7×7	0.3361	0.0062	$\approx 54\times$
256×256	15×15	0.3560	0.0216	$\approx 17\times$
512×512	3×3	1.3480	0.0068	$\approx 198\times$
512×512	7×7	1.3710	0.0247	$\approx 56\times$
512×512	15×15	1.4760	0.0918	$\approx 16\times$

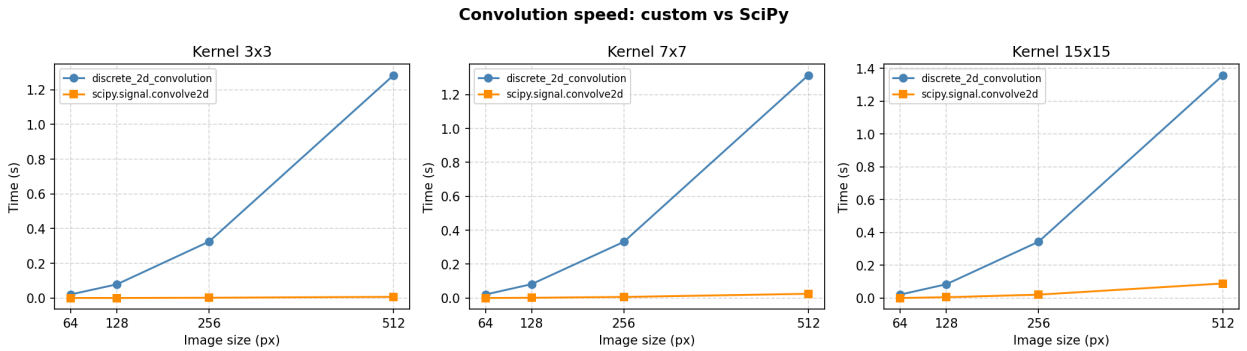

Figure 1: Execution time (seconds) as a function of image size for three kernel sizes. SciPy is consistently faster by one to two orders of magnitude.

The custom function uses two nested Python `for`-loops (one over each output pixel) for a total of $H \times W$ Python iterations. Inside each iteration, `np.sum(patch * kernel)` performs $K_h \times K_w$ floating-point operations in C, but the loop itself is interpreted by the Python runtime. The overall time complexity is $\mathcal{O}(H \cdot W \cdot K_h \cdot K_w)$, and the constant factor is large because Python loop overhead dominates for small kernels. In contrast, `scipy.signal.convolve2d` is implemented in compiled C and avoids Python-level loops entirely.

For a small kernel (for example 3×3), the inner NumPy work per Python iteration is tiny, so Python overhead is the bottleneck and the speedup of SciPy over the custom implementation is largest ($\approx 200\times$). As the kernel grows (for example 15×15), each Python iteration triggers more NumPy work, slightly amortising the overhead, and SciPy's own cost grows too, leading to a smaller but still substantial speedup ($\approx 16\times$).

Question 2: Blurring kernel

The chosen kernel is a Gaussian kernel of size 15×15 with standard deviation $\sigma = 3$ pixels, defined as:

$$K[m,n] = \frac{1}{Z} \exp\left(-\frac{m^2+n^2}{2\sigma^2}\right), \qquad Z = \sum_{m,n} \exp\left(-\frac{m^2+n^2}{2\sigma^2}\right), \tag{1}$$

where the normalisation constant Z ensures the kernel sums to 1, so that the overall image brightness is preserved. The kernel is visualised in Figure 2 and the blurring result in Figure 3.

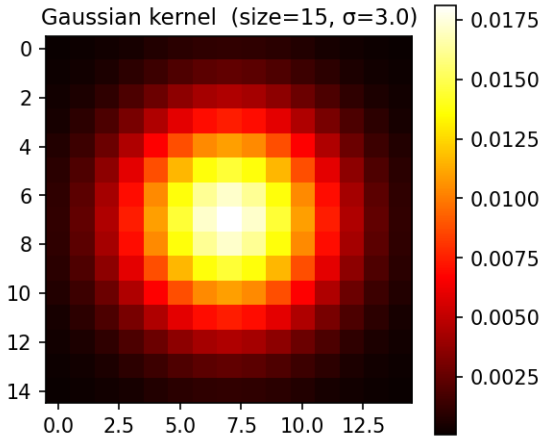

Figure 2: Gaussian kernel (15×15, $\sigma = 3$). Weights are highest at the centre and decay smoothly towards the edges.

Figure 3: Left: original image. Right: image after applying the 15×15 Gaussian blur kernel ($\sigma = 3$).

Convolving an image with a kernel replaces every pixel value with a weighted average of its neighbourhood. When the kernel weights are non-negative and sum to 1 (as in Equation (1)), each output pixel becomes a mixture of nearby input pixels, which has two immediate consequences. First, high-frequency detail is attenuated: sharp transitions (edges, fine textures) correspond to rapid spatial changes in pixel intensity. Averaging over a neighbourhood dampens these abrupt changes, reducing high-frequency energy and producing a smoother image. Second, low-frequency structure is preserved: regions of slowly varying intensity (large uniform areas, coarse shapes) are nearly constant across the averaging window, so their contribution to the output is barely changed.

The Gaussian weighting is preferable to a flat *box* (average) kernel because it gives more weight to pixels close to the centre and progressively less weight to distant ones. This smooth roll-off produces a more natural blur than a sharp rectangular window.

Question 3: Edge-detection kernels

The Sobel operator is used, a standard first-order derivative filter that combines a finite-difference gradient with Gaussian smoothing along the orthogonal axis. Two 3×3 kernels are defined:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

G_y (horizontal edges) approximates the partial derivative of the image intensity along the vertical axis ($\partial I/\partial y$). G_x (vertical edges) approximates the partial derivative along the horizontal axis ($\partial I/\partial x$). Both kernels are given in Equation 2.

Results

Figure 4 shows the original image alongside the absolute gradient responses of the two kernels.

Figure 4: Left: original image. Centre: horizontal edges ($|G_y * I|$). Right: vertical edges ($|G_x * I|$). Bright pixels indicate strong gradient responses.

As expected, G_y highlights the roof line, hood, and bumpers (predominantly horizontal boundaries), while G_x highlights the windshield pillars, wheel arches, and side panels (predominantly vertical boundaries).

Each Sobel kernel can be decomposed as the outer product of a smoothing vector and a differencing vector:

$$G_y = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^\top \qquad G_x = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^\top \tag{3}$$

The differencing vector computes a numerical first derivative: it subtracts pixel values on one side of a boundary from those on the other side. Where intensity is locally constant (flat region) the difference is zero; where intensity changes abruptly (edge) the difference is large. The smoothing vector averages over the perpendicular direction, which reduces sensitivity to noise while still localising the edge accurately. Taking the absolute value of the output captures both rising and falling transitions as bright ridges in the response map.

Question 6: MLP vs MobileNet on PathMNIST

Experimental setup

Both models are trained on PathMNIST using the fixed hyperparameters in `main_classification.py`. The AdamW optimiser with learning rate 10^{-3} and weight decay 5×10^{-4} is used, with a batch size of 128, and training for 15 epochs with CrossEntropyLoss. During training, random horizontal flips and random resized crops are applied as data augmentation.

Results

Table 2: Final test performance after 15 epochs.

Model	Best val. accuracy	Test accuracy
MLP	0.697	0.687
MobileNet	0.936	0.809

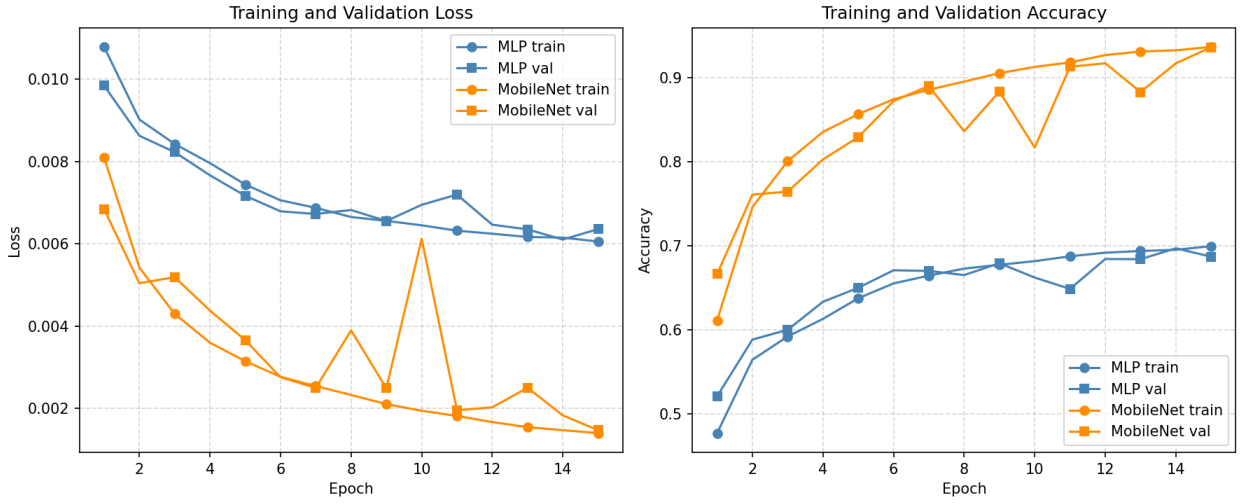

Figure 5: Training and validation loss (left) and accuracy (right) over 15 epochs. MLP: blue; MobileNet: orange. Circles: train, squares: validation.

MobileNet significantly outperforms the MLP on this task, achieving a best validation accuracy of **93.6%** compared to the MLP's **69.7%**. Several factors explain this performance gap:

Inductive bias. MobileNet's convolutional architecture encodes two critical assumptions that match the structure of image data: translation equivariance, whereby convolutional filters detect the same features (edges, textures, patterns) regardless of their position in the image (in contrast to the MLP, which must learn separate weights for each spatial location), and local connectivity, in which convolutions process small spatial neighbourhoods, exploiting the fact that pixels are more strongly correlated with nearby pixels than distant ones while the MLP treats the image as a flat vector, discarding all spatial structure.

Hierarchical feature learning. MobileNet's architecture builds a hierarchy of increasingly abstract features: early layers detect low-level patterns (edges, gradients), middle layers combine them into textures and object parts, and late layers capture high-level semantic concepts. The MLP, with only a shallow stack of fully-connected layers, struggles to learn such hierarchies from scratch.

Test performance gap. MobileNet's test accuracy (80.9%) is lower than its validation accuracy (93.6%), while the MLP shows a smaller gap (68.7% test vs 69.7% val). This suggests MobileNet may have slightly overfit to the validation set (or the test set has a different distribution), though it still substantially outperforms the MLP on both splits.

In summary, convolutional architectures like MobileNet are fundamentally better suited to image classification tasks than fully-connected MLPs, exploiting spatial structure and translation invariance to achieve superior performance with comparable parameter counts.

Problem 4

Question 1: UNet without skip connections

Architecture

Both models use `segmentation_models_pytorch` (smp) with a ResNet-18 encoder and a standard UNet decoder, trained from scratch on the retina vessel segmentation dataset.

- **UNet (with skip):** `smp.Unet(encoder_name=''resnet18'')`, the standard configuration where encoder feature maps are concatenated to the decoder at each resolution level.
- **UNet (no skip):** same architecture, but before passing the encoder features to the decoder all skip-connection feature maps (that is, every feature tensor except the bottleneck) are replaced with zero tensors of the same shape. This ensures the decoder architecture is identical; the only change is that skip information is withheld.

All other hyperparameters (AdamW optimizer, lr $= 10^{-4}$, wd $= 5 \times 10^{-4}$, batch size 2, 40 epochs, DiceCE loss) are held constant so that the comparison isolates the effect of skip connections.

Results

Table 3: Best validation Dice score for UNet vs. UNet without skip connections (40 epochs, retina vessel segmentation).

Model	Best val. Dice
UNet (with skip)	0.8103
UNet (no skip)	0.6106

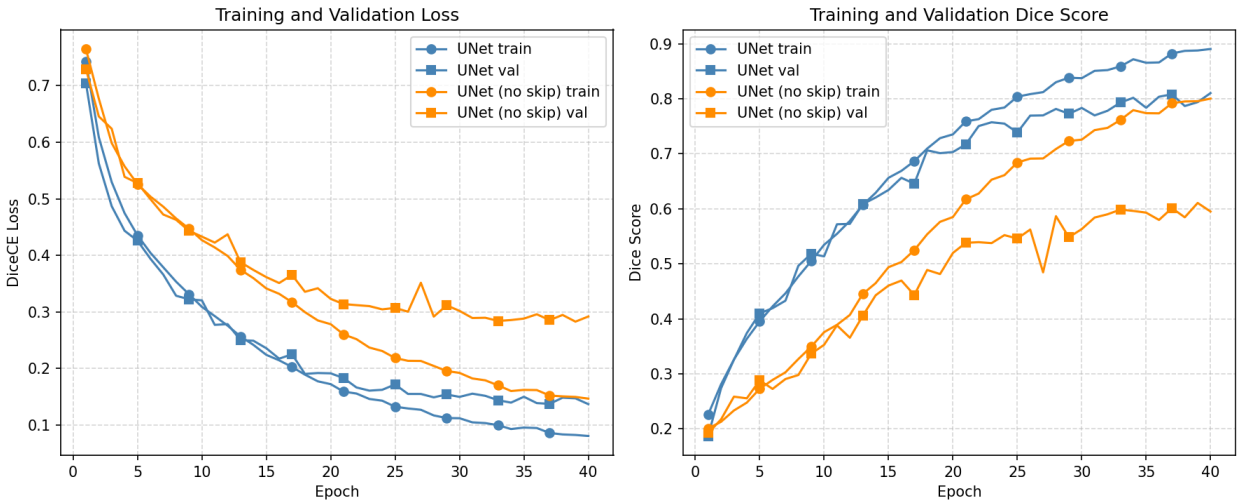

Figure 6: Training and validation Dice loss (left) and Dice score (right) for UNet with and without skip connections.

UNet with skip connections achieves a best validation Dice score of 0.8103, significantly outperforming the no-skip variant (0.6106) by ≈20 percentage points.

Without skip connections, high-frequency spatial information (vessel edges, thin capillaries, precise boundaries) is progressively lost as the encoder downsamples the input through multiple pooling operations, and the bottleneck representation cannot preserve fine-grained localization details. During upsampling, the decoder attempts to reconstruct spatial structure from this coarse representation alone, but the missing high-resolution details cannot be recovered through transposed convolutions, yielding blurry segmentations that correctly identify large vessels but fail to precisely delineate thin structures. Skip connections address this by directly concatenating encoder feature maps with decoder feature maps at each resolution level, giving the decoder both coarse semantic information from the bottleneck (what to segment) and fine spatial information from the encoder (where to segment). This dual pathway lets the decoder recover precise boundaries: encoder features supply localization cues (edges, textures) while decoder features supply semantic context (vessel vs. background classification). For retina vessel segmentation, thin capillaries (< 5 pixels wide) are critical diagnostic features, and the no-skip model, relying solely on the bottleneck, likely misses these or produces fragmented predictions.

Question 2: Effect of learning rate

Experimental setup

Model training uses `smp.Unet` (ResNet-18 encoder, same as Q1) with the Adam optimizer for 40 epochs using five learning rates: 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}. All other hyperparameters (batch size, weight decay, loss) are held constant.

Results

Table 4: Best validation Dice score per learning rate (40 epochs, Adam).

Learning rate	Best val. Dice
10^{-1}	0.2946
10^{-2}	0.8098
10^{-3}	**0.8383**
10^{-4}	0.8099
10^{-5}	0.3824

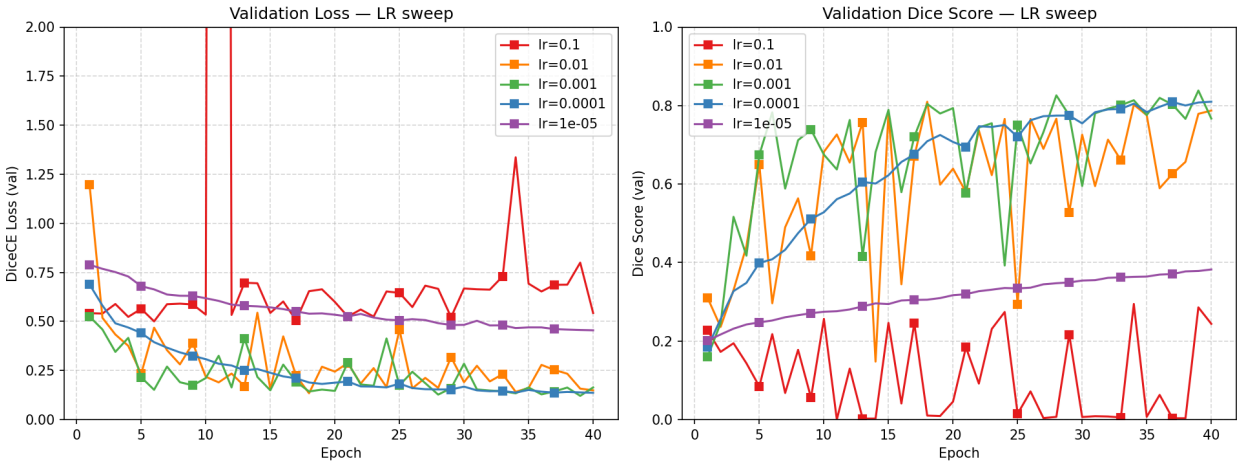

Figure 7: Validation DiceCE loss (left) and Dice score (right) for each learning rate across 40 epochs.

lr = 10^{-1}: divergence. The largest learning rate produces highly unstable training. The validation Dice score oscillates violently and never exceeds 0.30. Steps are so large that the optimizer overshoots good minima at every iteration.

lr = 10^{-2} and 10^{-4}: competitive but sub-optimal. Both learning rates converge reliably to similar final performance (≈ 0.81 Dice), despite taking different paths. 10^{-2} converges faster in early epochs but with slightly more noise; 10^{-4} converges more smoothly but at a slower pace, reaching the same value by epoch 40.

lr = 10^{-3}: best trade-off. The intermediate learning rate achieves the highest best validation Dice (0.8383), combining fast early-epoch progress with stable, monotone improvement throughout training.

lr = 10^{-5}: under-fitting or insufficient convergence. With very small steps the model barely trains in 40 epochs, achieving only 0.3824 Dice. The validation loss curve hardly decreases, indicating that the model is far from convergence.

Adam is sensitive to the learning rate. The optimal value (10^{-3}) balances step size against stability. Rates one order of magnitude higher cause divergence; rates one order of magnitude lower converge reliably but under-perform within the 40-epoch budget. Rates two or more orders of magnitude away from the optimum either diverge or fail to converge entirely.

Question 3: Data augmentation strategies

Strategies investigated

Four augmentation strategies are compared against a no-augmentation baseline, all applied exclusively to the training set.

- **Geometric**: random horizontal and vertical flips ($p = 0.5$ each) together with random rotations of up to ± 30 using reflect padding. Since retinal vessels have no preferred orientation, the model should be invariant to such transformations.
- **Photometric**: random brightness offset $\in [-40, 40]$ and contrast scaling $\in [0.7, 1.3]$ applied to the image only, mimicking variations in fundus camera calibration and illumination across patients.
- **Gaussian noise**: additive zero-mean Gaussian noise with standard deviation sampled uniformly from $[5, 25]$ applied to the image only. This simulates sensor noise and CCD read-out noise present in real fundus cameras, a source of variation orthogonal to global brightness or contrast changes.
- **Random zoom**: a random sub-region of scale $s \in [0.7, 1.0]$ is cropped and resized back to the original resolution (bilinear for image, nearest-neighbour for mask). This simulates different camera zoom levels and forces the model to be scale-invariant, addressing a source of variation not covered by the other three strategies.

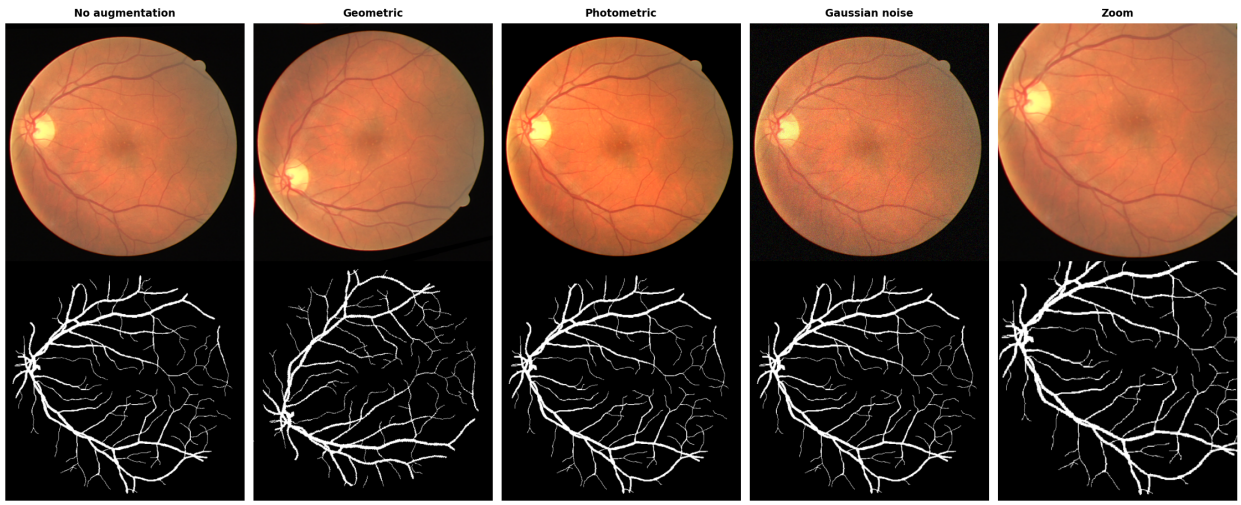

Figure 8: Example augmented training image and corresponding mask for each strategy (columns, left to right: no augmentation, geometric, photometric, Gaussian noise, zoom). Top row: RGB image. Bottom row: binary vessel mask.

Results

Table 5: Best validation Dice score per augmentation strategy (40 epochs).

Strategy	Best val. Dice
No augmentation	0.8057
Geometric	**0.8141**
Photometric	0.8117
Gaussian noise	0.7958
Random zoom	0.7787

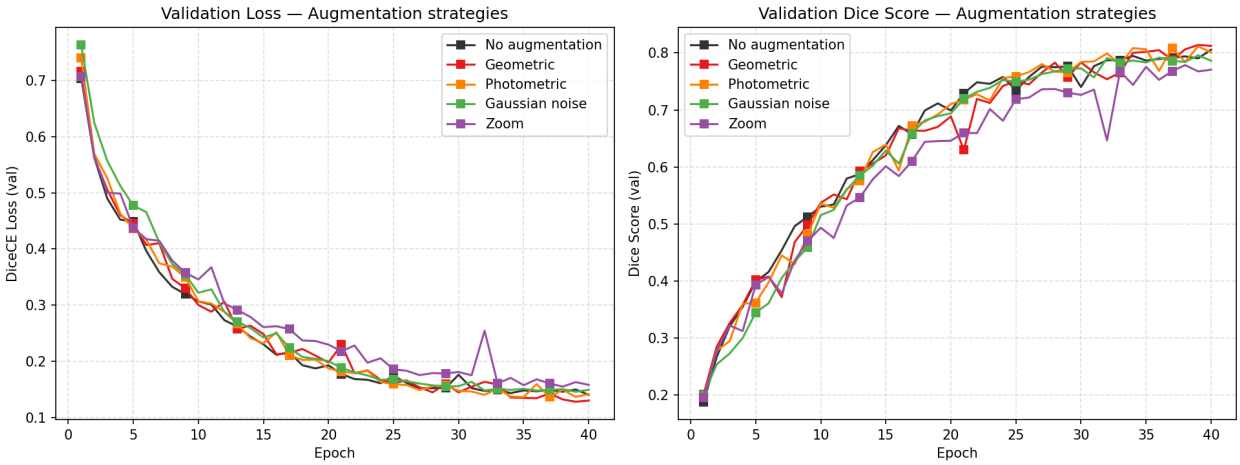

Figure 9: Validation DiceCE loss (left) and Dice score (right) for each augmentation strategy across 40 epochs.

The results reveal a clear split between augmentations that help and those that hurt on this small dataset (80 training images).

Geometric and photometric augmentations improve generalisation. Geometric augmentation achieves the best validation Dice (**0.8141**, +0.84 pp over the baseline), confirming that orientation invariance is a strong prior for retinal vessel segmentation: vessels have no preferred direction, so flips and moderate rotations generate genuinely new training samples. Photometric augmentation ranks second (0.8117, +0.60 pp), reflecting real inter-patient variation in fundus camera illumination and calibration. Both strategies add meaningful diversity without distorting the spatial structure of the labels.

Gaussian noise and zoom hurt performance. Gaussian noise (0.7958, −0.99 pp) and random zoom (0.7787, −2.70 pp) both perform below the no-augmentation baseline. Gaussian noise introduces per-pixel stochastic perturbations that partially obscure the fine texture cues the model relies on to detect thin capillaries; with only 80 training images, there are too few examples for the model to learn noise invariance reliably. Random zoom (scale $\in [0.7, 1.0]$) is particularly aggressive: cropping up to 30% of the image can remove entire vascular structures from the visible field, creating ambiguous training samples where vessels abruptly disappear at the crop boundary.

For small medical imaging datasets, conservative augmentations that respect the imaging geometry (flips, rotations) and simulate realistic acquisition variability (brightness, contrast) are most effective. Aggressive spatial or noise-based augmentations require larger datasets to be beneficial.

Question 4: Pretrained model fine-tuning

Setup

Two models are compared, trained under identical conditions (AdamW, lr $= 10^{-4}$, wd $= 5\times10^{-4}$, 40 epochs, DiceCE loss):

- **UNet (from scratch)**: `smp.Unet` with a ResNet-18 encoder initialised *randomly* (results from Q1).
- **ResNet18-UNet (fine-tuned)**: identical architecture, but the ResNet-18 encoder is initialised with ImageNet weights. The decoder is randomly initialised and trained from scratch in both cases.

The only difference is the encoder initialisation. Both models have the same number of trainable parameters.

Results

Table 6: Best validation Dice score: from scratch vs. fine-tuned.

Model	Best val. Dice
UNet (from scratch)	0.8103
ResNet18-UNet (fine-tuned)	**0.8296**

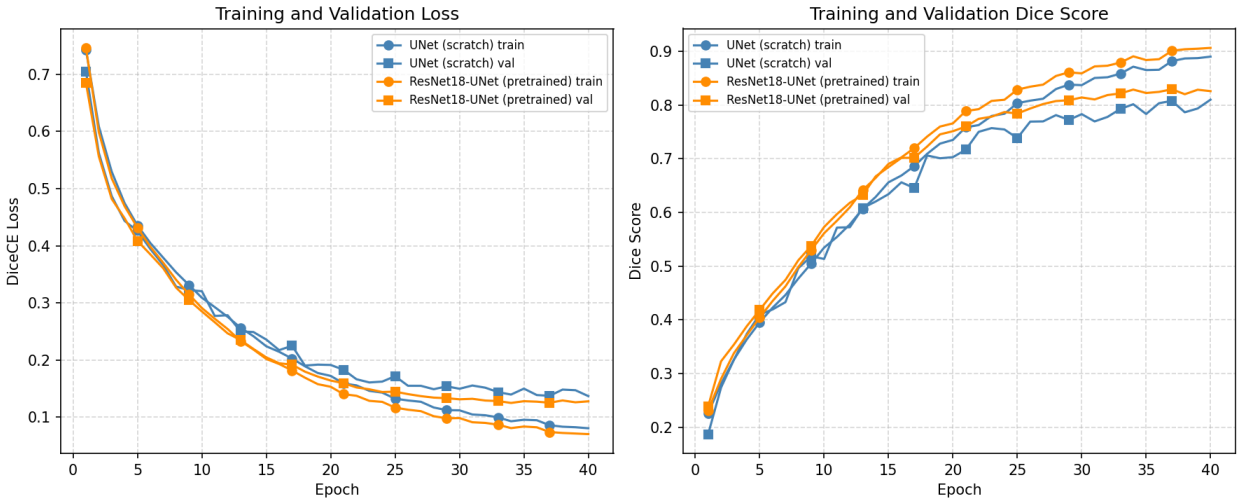

Figure 10: Training and validation DiceCE loss (left) and Dice score (right). Blue: UNet from scratch. Orange: ResNet18-UNet fine-tuned from ImageNet weights.

Fine-tuning the ImageNet-pretrained encoder improves the best validation Dice from 0.8103 to 0.8296 (+1.9 pp) with no additional parameters or training cost.

The pretrained encoder already extracts meaningful low-level features (edge detectors, gradient filters, texture responses) that are directly relevant to vessel detection. As a result, the fine-tuned model reaches higher Dice scores in the first few epochs and seems to keep that same advantage throughout the epochs. This effect might be stronger here since we are in a limited-data setting where learning features quickly is important.

ImageNet contains natural photographs, not fundus images. Despite this domain gap, low-level features (oriented edges, blob detectors) transfer well: retinal vessels are thin elongated structures whose detection relies on exactly such features. Higher-level semantic features (object parts, scene categories) are less transferable, which is why the decoder must still be trained from scratch.

Transfer learning is most beneficial when (1) the source dataset is large and diverse, (2) the target dataset is small, and (3) low-level feature structure is shared across domains. All three conditions hold here, which explains the consistent improvement over random initialisation.