Table of Contents

Computer Vision for CS & AI

Laboratory Manual — Student Edition

Fifteen two-hour sessions. One per week, each paired with a chapter of the textbook.

Before the first session

1. You need a Google account and nothing else. All work happens in **Google Colab** in a browser. Do not install Python locally for this course unless you want to.
2. Open each week's notebook and immediately choose **File → Save a copy in Drive.** Work in your copy. If you edit the shared original your work will not be saved.
3. Run cells in order with `Shift + Enter`. If something behaves oddly, use **Runtime → Restart and run all** before asking for help — it resolves most problems.

What you need each week

Week	Runtime	Installs	Downloads
1–10, 12–14	CPU is fine	none	none
11	**GPU** (Runtime → Change runtime type → T4)	one line, provided	your own annotated images
15	**GPU**	one line, provided	≈ 350 MB model, automatic

Almost every image used in this course ships inside `scikit-image`. Nothing downloads, so a slow connection or a blocked dataset link cannot stop you.

How labs are assessed

Each lab is marked out of 20 and the set forms your **lab portfolio**, worth 10% of the module, with a further 15% carried by the project work that grows out of Labs 11 to 14. Together the labs evidence **CLO4** (build applications) and **CLO5** (evaluate and optimise).

Marks are given for evidence, not for fluency. A reflection that quotes your own measured numbers scores well; an elegantly written one that could have been produced without running anything does not.

Submitting

Upload the executed notebook — **all cells run, outputs visible** — via `File → Download → Download .ipynb`. A notebook whose cells have not been run is marked as not submitted.

Using AI assistants

You may use them. Every block you did not write yourself must carry a `# ASSISTED` comment, and you must be able to explain it if asked. This is the same standard the textbook applies to itself: **say where things came from.** Unmarked assisted code is an academic-integrity matter, not a style issue.

If you get stuck

Work in pairs, and give it ten minutes before asking. Most errors in this course are one of four things: an array that is `uint8` where the code expects a float in $[0, 1]$, a missing `.copy()`, a row/column versus x/y mix-up, or a shape that is not what you assumed. Print `dtype`, `shape`, `min` and `max` before you print anything else.

Lab 0 — Getting Set Up (two hours)

Do this before Lecture 1 and before Lab 1. It is compulsory and it is not marked for correctness — only for completion. Students who skip it lose most of Lab 1 to setup problems that this session would have found.

Objective. Leave with a working environment, a dataset you can load, one image you have read and one you have written, results saved where you can find them again, and a self-test that says you are ready. **Setup.** A browser and a Google account. Nothing is installed on your laptop.

Time	Activity
0:00–0:15	**Colab, and the one habit that matters.** Open the Lab 0 notebook. Immediately choose **File → Save a copy in Drive** — if you skip this, nothing you do today is saved. Rename your copy `CV_Lab00_yourname`. Find **Runtime → Change runtime type** and note where it is; you will need GPU twice this semester and CPU every other week. Run the first cell.
0:15–0:30	**Check the toolchain.** Run the version cell. It prints the versions of `numpy`, `matplotlib`, `scikit-image`, `PIL` and `pandas`, and asserts each imports. Nothing needs installing — if an assertion fails, that is a real finding and you should report it rather than work around it.
0:30–0:55	**Load a dataset, three ways.** (a) From the bundled samples: `skimage.data.chelsea()` — this is what almost every lab uses, and it means no download can fail. (b) From a file you upload with the Colab file picker. (c) From a URL. For each, print `shape`, `dtype`, `min`, `max`. **Get into the habit of printing those four before anything else** — most errors in this course are visible in them.
0:55–1:15	**Mount your Drive and build the folder you will use all semester.** Run the mount cell and approve the permission prompt. Create `MyDrive/cv_course/` containing `data/`, `outputs/` and `notebooks/`. List it to confirm. Everything you produce for the rest of the course goes in `outputs/`.

Time	Activity
1:15–1:40	**Write something, then prove you wrote it.** Convert your image to grayscale, save it to `outputs/` as PNG, then **read it back from disk** and assert the shape matches. Saving without verifying is how people lose a week's work. Then compute five simple measurements (height, width, channels, mean, standard deviation), save them as `outputs/lab00_results.csv`, load the CSV back and print it.
1:40–1:52	**Make it reproducible.** Set `np.random.seed`, generate random numbers, restart the runtime, and confirm you get the same numbers. Then run **Runtime → Restart and run all** end to end. A notebook that only works when cells are run out of order is not finished, and this is the single most common reason a submitted lab scores zero.
1:52–2:00	**Self-test and submission dry run.** Run the final cell. It checks all eight items and prints PASS or FAIL for each. Fix anything red. Then practise submitting: `File → Download → Download .ipynb`, and upload it to the course page. Doing this once now means the real deadline is not the first time you try.

Deliverable. Your executed notebook with the self-test printing **8/8 PASS**, plus the file `lab00_results.csv` in your Drive.

Expected results. `chelsea()` has shape `(300, 451, 3)`, dtype `uint8`, with `min` 0 and `max` 231 — note that an 8-bit image need not actually reach 255. Your grayscale save-and-reload must return shape `(300, 451)`. The CSV must contain five rows and survive a round trip.

Common pitfalls.

- Working in the shared notebook instead of your own copy, so nothing saves. Check the title bar says Copy of… or your own name.
- Expecting `skimage` float images to be in `0–255`. After `rgb2gray` they are floats in `0–1`. Print `min` and `max` and you will never be caught by this.
- Mounting Drive but writing to `/content/` instead of `/content/drive/MyDrive/…`. Files in `/content/` vanish when the runtime recycles.
- Assuming a cell ran because it looked right. If there is no output, it did not run.

If something fails. Bring it to the first ten minutes of Lab 1 and we will fix it there. Do not spend the evening on it alone.

Lab 1 — Images as Numbers (two hours)

Objective. Show experimentally that raw-pixel comparison fails, and that a single normalisation step repairs it. **Evidences CLO1 and CLO4. Setup.** Google Colab, CPU runtime. No installation: `numpy`, `matplotlib` and `scikit-image` are preinstalled. All photographs come from `skimage.data`, so nothing is downloaded and no dataset link can break.

Time	Activity
0:00–0:10	Open the notebook, `Save a copy in Drive`, run the setup cell. Confirm `skimage` imports.
0:10–0:30	**Exercise 1 — an image is an array.** Load `data.chelsea()`. Print `shape`, `dtype`, `min`, `max`. Print the 10×10 eye patch as integers and reproduce Figure 1.1(c). Then print a patch from the blurred background and compare the two.
0:30–0:45	**Exercise 2 — channels.** Split R, G, B and display them. Reverse the channel order to produce the RGB/BGR bug deliberately, describe the symptom in one sentence, then fix it in one line.
0:45–1:10	**Exercise 3 — the nuisance factors.** Two functions are given (illumination, noise). Write the other four: viewpoint, scale, occlusion, deformation. Regenerate Figure 1.4 with your own numbers and rank the factors by pixel damage.
1:10–1:40	**Exercise 4 — break a classifier.** Build the nearest-template recogniser over ten gallery images. Confirm 100% on the originals. Sweep brightness $\alpha \in \{1.0, 0.8, 0.6, 0.4, 0.2\}$ and record accuracy at each. It collapses to about 10%. Then add one normalisation line, $q \leftarrow (q - \mu_q)/\sigma_q$ applied to query and templates alike, and re-run the sweep.
1:40–1:50	**Exercise 5 — how little is enough.** Reduce resolution until the subject is unidentifiable. Record your threshold and compare with three classmates.
1:50–2:00	**Exercise 6 — ethics checkpoint.** Print accuracy at $\alpha = 0.6$ overall and for two subgroups of the gallery. Overall reads 20%; the groups read 40% and 0%. Write two sentences connecting this to Buolamwini and Gebru (2018).

Deliverable. The executed notebook plus a 200-word reflection: given Exercise 4, what property must a good image representation have? Cite your own numbers from the brightness sweep.

Expected results. 100% before dimming, about 10% at $\alpha = 0.2$, and back to 100% after normalisation. If your normalised version does not recover, check that you normalised the templates as well as the query.

Common pitfalls. Forgetting `.copy()` in the occlusion function and destroying the original array; comparing `uint8` with `float` images without converting; normalising the query only.

Lab 2 — Light, Colour and Histograms (two hours)

Objective. Implement histogram equalisation from first principles and find the conditions under which it helps and under which it does harm. **Evidences CLO1 and CLO4. Setup.** Colab, CPU. No installs.

Time	Activity
0:00–0:15	Load a photograph, convert to grayscale, plot the image beside its histogram. Identify from the histogram alone whether it is over-, well- or under-exposed, then check against the image.
0:15–0:40	**Build the histogram.** Write `histogram(img)` with a loop and no library call, then verify against `np.bincount`. Normalise to $p[v]$ and accumulate to $c[v]$. Assert $c[L-1] = 1$ to within floating-point tolerance.
0:40–1:05	**Equalise.** Implement $T(v) = round\,((L-1)c[v])$ as a 256-entry lookup table and apply it by indexing, not by looping over pixels. Time both approaches and report the speed-up.
1:05–1:30	**Where it helps.** Apply your equaliser to the same photograph at gamma 0.45, 1.0 and 2.2. Plot a 3×2 grid of image-and-histogram pairs before and after. Which of the three improved most?
1:30–1:50	**Where it hurts.** Apply it to an already well-exposed photograph and to one with a large flat background. Quantify the damage: report the standard deviation of a flat patch before and after. Explain the increase.
1:50–2:00	**Ethics checkpoint.** Read the box on imaging pipelines calibrated for lighter skin. In three sentences, connect it to what you just saw: a global transform chosen for an assumed average will mistreat whatever the average was not built from.

Deliverable. The notebook, plus a short table reporting flat-patch standard deviation before and after equalisation for all five test images, and one paragraph stating when you would not use this technique.

Expected results. Large visible improvement on the $\gamma = 2.2$ under-exposed image; almost no change on the well-exposed one; a measurable rise in flat-patch noise on the image with a large background.

Common pitfalls. Building the lookup table but then applying it with a Python loop and wondering why it is slow; forgetting that `skimage` float images live in $[0, 1]$ while the equations here assume $[0, 255]$.

Lab 3 — Filtering and Edges (two hours)

Objective. Write convolution yourself, feel how slow the naive version is, and use that to motivate why the next six chapters run on GPUs. **Evidences CLO1 and CLO3. Setup.** Colab, CPU. No installs.

Time	Activity
0:00–0:20	**Convolution by hand.** Write `convolve2d(img, kernel)` with four nested loops. Test it on the 3×3 patch from Worked example 3.1 and assert the answer is exactly 240. Do not proceed until it is.
0:20–0:35	**Verify and time.** Compare your output against `scipy.signal.convolve2d` on a real photograph; assert the maximum absolute difference is below 10^{-9}. Time both. Expect your version to be 50–200× slower, and record the actual factor.
0:35–1:00	**A kernel zoo.** Apply blur, sharpen, emboss, K_x and K_y to one photograph. For each, state in one sentence what the weights do and why they sum to what they sum to.
1:00–1:25	**Gradients.** Compute I_x, I_y, then $\| \nabla I \|$ and θ. Display the magnitude as a heatmap over the photograph and the orientation as hue. Find a region where the magnitude is high but the orientation is unstable, and explain it.
1:25–1:45	**Separability.** Implement the 9×9 Gaussian both directly and as two 1-D passes. Assert the outputs agree, then time both and check the measured speed-up against the $4.5\times$ predicted in Worked example 3.2.
1:45–2:00	**Thresholds are a choice.** Run Canny at three threshold pairs on a medical or industrial image. Note which structures appear and vanish. Write three sentences on who should be allowed to pick that threshold and on what evidence.

Deliverable. The notebook with your timing table (naive vs SciPy, direct vs separable), and one paragraph answering: your convolution is correct and 200× too slow — what specifically makes the library version fast?

Expected results. Exact agreement with SciPy; a naive-loop penalty of two orders of magnitude; a separable speed-up close to, but a little below, the theoretical $4.5\times$ because of Python overhead.

Common pitfalls. Forgetting to flip the kernel and then being surprised that your result matches correlation rather than convolution — which is the point of §3.2, so say which one you implemented; mishandling borders and comparing against SciPy's `mode='same'` without matching the padding.

Lab 4 — Features and Matching (two hours)

Objective. Build a panorama from two photographs, then break it, and measure what RANSAC is worth. **Evidences CLO3. Setup.** Colab, CPU. `scikit-image` provides ORB and RANSAC; no installs.

Time	Activity
0:00–0:15	**Corners first.** Compute I_x, I_y, form M over a 5×5 window, and evaluate R per pixel. Display R as a signed heatmap. Confirm by eye that edges are negative and corners positive.

Time	Activity
0:15–0:30	Verify your R against the three matrices of Worked example 4.1. All three must reproduce to the printed values before you go on.
0:30–0:55	**Detect and describe.** Run ORB on two overlapping views. Plot keypoints, then plot matches. Apply the ratio test and plot again. Record how many matches survive.
0:55–1:25	**Fit with and without RANSAC.** Estimate the homography by plain least squares over all matches, then by RANSAC. Report mean reprojection error for both. Expect roughly 13 px against 0.4 px — a factor near 37.
1:25–1:45	**Stitch.** Warp and blend into a panorama. Inspect the seam: where it is visible, say whether the cause is geometry, exposure, or both.
1:45–2:00	**Break it.** Re-run the whole pipeline on a low-texture pair — a blank wall, a clear sky. It will fail. Using R and the structure tensor, explain precisely why it had to fail, and predict N from Worked example 4.2 for the inlier fraction you observe.

Deliverable. The panorama, the error table for least squares vs RANSAC, and a paragraph explaining the failure case in terms of eigenvalues rather than in terms of "not enough detail".

Expected results. Least squares is dragged off by a handful of wrong matches; RANSAC ignores them. The blank-wall case yields few keypoints, mostly noise, and a homography that is meaningless even when it is numerically returned.

Common pitfalls. Mixing (x, y) and (row, col) conventions between `skimage` and your own code — the single most common source of a panorama that warps the wrong way; treating a returned homography as a success without checking the inlier count.

Lab 5 — Learning a Classifier (two hours)

Objective. Write softmax regression and its gradient from scratch, with no framework, and see all three learning-rate regimes with your own eyes. **Evidences CLO1 and CLO4. Setup.** Colab, CPU. `numpy` and `scikit-learn` only — no deep learning library in this lab, deliberately.

Time	Activity
0:00–0:15	Load the digits dataset, inspect shapes, and split train/test. Predict the initial loss before running anything: it must be near $\log 10 = 2.303$.
0:15–0:35	**Forward pass.** Implement `softmax(S)` with the max-subtraction trick, then cross-entropy. Verify on the hand example of Worked example 5.1 that you reproduce $p_1 = 0.690$ and $L = 0.371$ exactly.

Time	Activity
0:35–1:00	**Backward pass.** Implement the gradient $p - e_y$ and one update step. Reproduce the printed update: $w_1 = 0.562$, $w_2 = -0.269$, $b = 0.131$, and a loss that falls to 0.354. Then check your gradient numerically against a finite difference on a random weight.
1:00–1:25	**Train.** Run the loop for 4,000 steps at $\eta = 0.5$ with minibatches of 128. Plot the loss. Report test accuracy; about 92% is expected.
1:25–1:45	**Three regimes.** Re-run at $\eta = 2 \times 10^{-4}$, $\eta = 0.5$ and $\eta = 60$. Plot all three curves on one axis and reproduce Figure 5.2, including the run that reaches `NaN`. Say in one sentence how you would tell "too small" from "broken" if you only had the first fifty iterations.
1:45–2:00	**Look at what it learned.** Reshape each row of W to image shape and display it. Connect what you see to Worked example 5.2: why must these be blurred?

Deliverable. The notebook, the three-regime plot, and a paragraph on the ceiling: you reached about 92% and a larger learning rate will not fix the rest — what will, and why?

Expected results. Initial loss ≈ 2.30; final test accuracy $\approx 92\%$; finite-difference and analytic gradients agreeing to about 10^{-6}; a `NaN` at the largest learning rate.

Common pitfalls. Omitting the max subtraction and getting `inf` from `exp`; averaging the gradient over the batch in one place but not the other, so the effective learning rate is off by a factor of the batch size.

Lab 6 — From Filters to CNNs (two hours)

Objective. Watch edge detectors appear from data alone, and confirm the parameter argument with your own counts. **Evidences CLO1 and CLO4. Setup.** Colab. `scikit-learn` provides FastICA; no downloads — the patches come from `skimage.data`.

Time	Activity
0:00–0:15	**Count first.** Before writing anything, compute P_{fc} and P_{conv} for the layer in Worked example 6.1 by hand. Then write a function that returns both and check it against your arithmetic.
0:15–0:30	**Output sizes.** Implement H_{out} for arbitrary k, p, s. Tabulate the spatial size through a five-layer stack and confirm it matches what you predicted.
0:30–1:05	**Learn filters from photographs.** Sample 24,000 patches of 11×11 from five photographs, subtract each patch mean, and fit 24 ICA components. Display them as a tile grid. You should see oriented edges and bars that nobody specified.

Time	Activity
1:05–1:20	**Starve it.** Repeat with 300 patches. Display side by side with the previous panel and describe the difference in one sentence.
1:20–1:45	**Use them.** Convolve a photograph with three of your learned filters, apply ReLU and max-pool, and repeat for three layers. Display the feature maps at each depth, reproducing Figure 6.2. Report the spatial size at each layer and check it against your formula from the second block.
1:45–2:00	**Interpretation, carefully.** Pick one channel and write what you think it responds to. Then find an image where your claim fails. Write two sentences on why single-channel interpretation is a hypothesis rather than a reading.

Deliverable. The filter tile grids for 24,000 and 300 patches, the three-depth feature maps with sizes labelled, and your falsified interpretation claim.

Expected results. Clean oriented filters at 24,000 patches, visible noise at 300; feature maps that grow less picture-like with depth; predicted and actual spatial sizes agreeing exactly.

Common pitfalls. Forgetting to subtract the patch mean, which lets ICA spend components on brightness instead of structure; displaying filters without per-filter normalisation, so the low-energy ones look blank.

Lab 7 — Generalisation (two hours)

Objective. Take a model that overfits, fix it, and prove the fix is real rather than lucky. **Evidences CLO4 and CLO5. Setup.** Colab, CPU. `numpy`, `scipy.ndimage`, `scikit-learn`.

Time	Activity
0:00–0:15	Train the provided model on 100 images. Plot training and validation loss together. Identify the epoch at which they part company and state what that gap means.
0:15–0:35	**Error bars first.** Before trying any fix, compute SE for your validation accuracy. Decide now what size of improvement you would need to believe a change helped. Write that number down before you see any results.
0:35–1:05	**Augmentation.** Implement rotation $\pm 12^{\circ}$ and shifts of ± 1 px. Expand the training set fourfold and retrain. Display a grid of augmented versions of one image to confirm the transforms are sane.
1:05–1:30	**Five seeds, not one.** Re-run both conditions across five seeds at training sizes $50, 100, 200, 400, 800, 1400$. Plot means with ± 1 s.d. bands and reproduce Figure 7.3.
1:30–1:45	**Read it horizontally.** Estimate how many real images the augmentation was worth at $n = 100$, by finding where the unaugmented curve reaches the augmented value. Report the number.

Time	Activity
1:45–2:00	**Ethics checkpoint.** Augmentation multiplied your images. State clearly what it did not multiply, and give one example of a group that a rotation cannot conjure into a dataset.

Deliverable. The banded plot, the number you wrote down in the second block compared against what you actually observed, and one paragraph: did the augmentation help by more than its error bars, and at which training sizes?

Expected results. A clear gap at small n that narrows as n grows; at the largest sizes the bands may overlap, meaning the benefit is no longer demonstrated.

Common pitfalls. Augmenting the validation set as well as the training set, which quietly invalidates the comparison; changing two things at once and being unable to attribute the improvement.

Lab 8 — Transfer Learning (two hours)

Objective. Measure what a pretrained representation is worth, and find the conditions under which it is worth nothing. **Evidences CLO4 and CLO5. Setup.** Colab. `scikit-learn` and `skimage` only; the source and target "datasets" are patches from bundled photographs, so nothing is downloaded.

Time	Activity
0:00–0:15	**Predict first.** Using Worked example 8.1, compute examples-per-parameter for both arms at $n = 40, 160, 400$. Write down where you expect the gap to be largest before running anything.
0:15–0:40	**Build the source task.** Sample 32×32 patches from six texture and greyscale photographs; standardise each patch. Train a 128-unit hidden layer to predict which photograph a patch came from. Report source accuracy.
0:40–1:00	**Freeze and transfer.** Extract the hidden activations as features. Train a linear head on four different photographs at several labelled-set sizes.
1:00–1:20	**The fair comparison.** Train a linear classifier directly on the raw 1,024 pixels at the same sizes. Both arms are linear, so the only thing that differs is the representation — say why that matters for the claim you are about to make.
1:20–1:40	Plot both curves with a chance line at $1/4$. Reproduce Figure 8.2 and report the largest gap and where it occurs. Compare with your prediction from the first block.
1:40–2:00	**The negative result.** Repeat the entire comparison on 8×8 digits. The benefit disappears. Explain it using D/d, not by saying the method "did not work".

Deliverable. Both curves, the prediction-versus-outcome comparison, and a paragraph stating the condition under which you would expect transfer to help a future project of your own.

Expected results. A gap of up to about 10 points on the texture task, with the scratch curve nearly flat; no measurable gap on digits.

Common pitfalls. Letting target images leak into the source task, which inflates the transfer arm; comparing frozen-feature transfer against a deeper scratch model and attributing the difference to transfer rather than capacity.

Lab 9 — Architectures and Depth (two hours)

Objective. Measure the accuracy–latency frontier yourself, and see the vanishing-gradient product rather than being told about it. **Evidences CLO4 and CLO5. Setup.** Colab, CPU is sufficient. `scikit-learn` and `numpy`.

Time	Activity
0:00–0:20	**The product, numerically.** Simulate a stack of L Jacobians with typical magnitude 0.8 and plot gradient scale against L on a log axis. Reproduce the three numbers in Worked example 9.1. Then add a skip route and plot both.
0:20–0:45	**Depth without skips.** Train networks of increasing depth on the same data with the same budget. Plot test accuracy against depth. Find the depth beyond which accuracy falls, and state why this is not an overfitting result.
0:45–1:15	**Measure latency honestly.** For six model sizes, record test accuracy, parameter count, and inference time per image averaged over 30 runs. Discard the first run — say why in one sentence.
1:15–1:35	**Build the frontier.** Plot accuracy against latency with marker area proportional to parameters. Compute the Pareto set programmatically: keep a point if no other point is both faster and more accurate. Reproduce Figure 9.3.
1:35–1:50	**Choose under a constraint.** You are given a budget of 50 µs per image. Name the model you would ship and justify it from your own plot, not from the largest accuracy in the table.
1:50–2:00	**Batch norm at test time.** Run one model with batch statistics at evaluation instead of running averages. Report the accuracy change and explain the mechanism.

Deliverable. The frontier plot with your Pareto set marked, the depth-versus- accuracy curve, and a one-paragraph backbone recommendation for your project with the constraint stated first.

Expected results. Gradient scale falling by six orders of magnitude by $L = 56$; a measurable accuracy drop at the deepest plain models; a frontier on which the largest model is usually not the right choice.

Common pitfalls. Timing a single forward pass and reporting the warm-up cost; comparing latency across runs while Colab reassigns your machine.

Lab 10 — The Detection Problem (two hours)

Objective. Implement IoU, NMS and AP in NumPy, verify against a library, and find the case where NMS must be wrong. **Evidences CLO2 and CLO5. Setup.** Colab, CPU. `numpy` and `matplotlib` only — no detector is trained in this lab, deliberately: the metric is the subject.

Time	Activity
0:00–0:20	**IoU by hand, then in code.** Compute Worked example 10.1 on paper. Then write `iou(a, b)` and assert it returns 0.279 to three decimals. Add tests for disjoint boxes (must be 0), identical boxes (must be 1) and containment.
0:20–0:40	**Visualise the threshold.** Draw box pairs at IoU 0.3, 0.5, 0.75 and 0.9 over a photograph. Write one sentence on how tight 0.9 actually is, and what that means for annotation consistency.
0:40–1:05	**NMS.** Implement it: sort by score, keep the top box, discard overlaps above τ_{nms}, repeat. Run on a scene with 18 duplicate boxes and confirm it reduces to 4.
1:05–1:25	**Where NMS must fail.** Construct two genuinely overlapping objects at IoU 0.31. Run NMS at 0.5 and observe one real object suppressed. State the trade-off: every threshold converts one error type into the other.
1:25–1:50	**Precision, recall, AP.** Sort detections by confidence, accumulate TP and FP, build the PR curve, apply the monotone envelope and integrate. Compute $AP_{0.50}$ and $AP_{0.75}$, then sweep ten thresholds for the COCO average. Reproduce Figure 10.5.
1:50–2:00	**Threshold as policy.** For a reversing car and for a tumour screen, state which of a false positive and a false negative is worse, and which way you would move the confidence threshold. Name who should sign off on that choice.

Deliverable. Your NumPy implementations with the assertions passing, both PR curves, the ten-threshold sweep, and the policy paragraph.

Expected results. $IoU = 0.279$; NMS $18 \to 4$; $AP_{0.50} \approx 0.74$ against $AP_{0.75} \approx 0.39$.

Common pitfalls. Forgetting $\max(0, \cdot)$ and getting negative intersections for disjoint boxes; matching each ground-truth object more than once, which silently inflates recall.

Lab 11 — Detectors in Practice (two hours)

Objective. Take a detection project end to end — annotate, train, evaluate, and look honestly at the failures. **Evidences CLO2. Setup.** Colab with a GPU runtime (`Runtime → Change runtime type → T4`). One install line, pre-written in cell 1: `pip install ultralytics supervision` (about 40 seconds). Ultralytics software uses AGPL-3.0, while some commercial deployments may require an Enterprise licence. Review the current Ultralytics licensing terms before publishing or deploying your work.

Time	Activity
0:00–0:20	**Annotate.** In teams, label about 100 images of a single class in Roboflow or CVAT. Agree a written rule for edge cases before starting — occluded objects, objects touching the frame edge, ambiguous instances.
0:20–0:30	**Measure your own agreement.** Two students label the same 10 images independently. Compute mean IoU between the two sets. Anything below about 0.7 means your rule was not specific enough; fix the rule, not the labels.
0:30–0:45	Export in the expected format, inspect the class-balance histogram, and split train/val without letting near-duplicate frames straddle the split.
0:45–1:20	**Train.** Fine-tune a small detector for a fixed epoch budget. Watch the loss curves separate into classification and localisation components; say which one is limiting you.
1:20–1:40	**Evaluate.** Report $AP_{0.50}$ and $AP@[.5 : .95]$ using the metric code you wrote in Lab 10, not only the library's number. If they disagree, find out why — that investigation is the exercise.
1:40–1:55	**Failure gallery.** Assemble the ten worst false positives and ten worst false negatives. Group them by cause: small objects, occlusion, unusual lighting, label error. Count how many are actually annotation mistakes.
1:55–2:00	**Ethics checkpoint.** Who labelled your data, how long did it take, and what would that cost at a fair wage? Write two sentences on annotation labour.

Deliverable. The trained model's metrics, the inter-annotator IoU, the grouped failure gallery, and the count of failures that were really label errors.

Expected results. Modest AP from 100 images — that is expected and is the point. Typically a substantial fraction of apparent model errors turn out to be annotation errors.

Common pitfalls. Near-duplicate frames in both train and validation, which inflates your score badly; changing the annotation rule partway through and producing an inconsistent dataset.

Lab 12 — Segmentation (two hours)

Objective. Show that the headline overlap score hides what a segmentation is actually used for. **Evidences CLO4 and CLO5. Setup.** Colab, GPU helpful but not required. `skimage`, `numpy`; a small U-Net is provided.

Time	Activity
0:00–0:15	**Metrics first.** Implement IoU and Dice for masks. Verify the identity $Dice = 2\,IoU/(1 + IoU)$ on ten random mask pairs to within 10^{-12}. Reproduce Worked example 12.1.

Time	Activity
0:15–0:35	**A baseline worth beating.** Segment the coins image by Otsu thresholding plus morphological cleanup — the Chapter 2 method. Record IoU and Dice. Every learned model today must be compared against this number.
0:35–1:00	**Boundary IoU.** Implement it: dilate the true boundary by $d = 5$ and restrict the comparison to that band. Recompute for your threshold baseline. Note how much lower it is than the global score.
1:00–1:30	**Encoder–decoder, with and without skips.** Run the provided model in both configurations. Report global IoU and boundary IoU for each, and reproduce Figure 12.3.
1:30–1:50	**Which number would you publish?** The global scores differ by about four points; the boundary scores by about twenty-six. Write a short paragraph choosing what to report for a use case you name, and defend it.
1:50–2:00	**Ethics checkpoint.** A segmentation model validated on one hospital's scanner is deployed on another's. Using what you just measured about boundaries, describe one concrete way this fails and one sentence that should appear in the model's documentation.

Deliverable. The metric table (threshold baseline, no skips, with skips × global and boundary), the identity check, and your reporting-choice paragraph.

Expected results. Boundary IoU markedly below global IoU for every method; the skip/no-skip gap several times larger under the boundary metric.

Common pitfalls. Computing Dice and calling it IoU; evaluating on upsampled masks so that interpolation quietly smooths your boundaries before they are scored.

Lab 13 — Motion and Tracking (two hours)

Objective. Estimate flow and score it against known truth, then cause an identity switch on purpose and diagnose it. **Evidences CLO3. Setup.** Colab, CPU. `skimage.registration` provides the flow; frames are built from bundled photographs, so no video download is needed.

Time	Activity
0:00–0:20	**Make truth.** Warp a photograph by a known 6° rotation about its centre. Derive the ground-truth flow analytically from the transform. Plot the field and confirm it circulates with a near-stationary centre. Compose the transform as translate–rotate–translate and check the order: reversed, you get a translation and will not notice.

Time	Activity
0:20–0:45	**Estimate and score.** Run dense optical flow. Compute mean endpoint error against your analytic truth; expect roughly 1.5 px. Visualise the field as arrows and as a hue-for-direction map, reproducing Figure 13.2.
0:45–1:05	**The aperture problem.** Construct a synthetic vertical edge translating diagonally. Show numerically that the recovered flow captures the horizontal component and not the vertical. Compute the structure tensor and relate its eigenvalues to Worked example 4.1.
1:05–1:35	**Tracking by detection.** Implement greedy IoU association across frames. Run it on two objects crossing at different speeds, with the detector merging boxes when they overlap above 0.45. Print which true object each track is on at every frame.
1:35–1:50	**Find the switch.** Identify the frame where identity swaps. Reproduce the distance arithmetic of Worked example 13.3 and confirm your tracker did exactly what the numbers predicted.
1:50–2:00	**Fix one, break the other.** Propose one change — a motion model, an appearance feature, a longer coasting window — implement or describe it, and say what new failure it introduces.

Deliverable. The endpoint error number, the aperture-problem demonstration, your per-frame identity table with the switch marked, and the proposed fix with its cost.

Expected results. EPE near 1.5 px; recovered flow along an edge resolving only the gradient-aligned component; an identity switch on the frame immediately after the merge.

Common pitfalls. Reversing the transform composition and silently producing a translation; comparing detection indices across frames as if they were stable identities — they are not, which is the entire lesson.

Lab 14 — Evaluation and Error Analysis (two hours)

Objective. Audit your own project model until the single accuracy number stops being the thing you trust. **Evidences CLO5. Setup.** Colab. Bring the model and validation set from your project.

Time	Activity
0:00–0:15	**Confusion matrix.** Build it for your own model and display it normalised by row. Identify the single largest off-diagonal cell and name the confusion in words.
0:15–0:35	**Error bars.** Compute the Wilson interval for your headline accuracy. State the smallest improvement you could detect with your current validation-set size. If that number is larger than the gains you have been reporting, say so plainly.

Time	Activity
0:35–1:05	**Slice it.** Define at least three meaningful groups — lighting, object size, source device, demographic if applicable and ethically appropriate. Report accuracy and its interval per group, plus the worst-group score. Reproduce the structure of Worked example 14.1 with your own numbers.
1:05–1:25	**Failure gallery.** Collect your twenty worst errors, display them, and group them by cause. Count how many are label errors rather than model errors.
1:25–1:45	**Optimise and re-measure.** Quantise or prune the model. Record accuracy, worst-group accuracy, size and latency before and after. Check specifically whether compression cost the worst group more than the average — it usually does.
1:45–2:00	**Write the model card.** One page: intended use, out-of-scope use, training data and its provenance, metrics overall and per group with intervals, known failure modes, and the date. This is a graded deliverable, not a formality.

Deliverable. The model card, the per-group table with intervals, the failure gallery with causes counted, and the before/after compression table.

Expected results. A worst-group score meaningfully below the mean; at least one group whose interval is too wide to conclude anything; some fraction of "model errors" that are really annotation errors.

Common pitfalls. Defining groups after seeing the results, which turns an audit into a search for a flattering split; reporting per-group means without intervals and over-reading a group of twelve images.

Lab 15 — Transformers, and What You Owe (two hours)

Objective. Look inside an attention map, then close the loop on the question Chapter 1 opened. **Evidences CLO4 and CLO5. Setup.** Colab, GPU runtime. One install line in cell 1: `pip install timm`. A pretrained ViT is downloaded once, about 350 MB.

Time	Activity
0:00–0:15	**Count before you load.** Compute N for $P = 16$ and $P = 8$ at 224^2, and the size of the attention matrix for each. Predict which will fit in the session's memory, then check.
0:15–0:35	**Patchify.** Split a photograph into 16×16 patches and display the grid. This is literally the model's input; note that patch boundaries cut objects arbitrarily and that nothing tells the model two adjacent patches are adjacent except a learned positional embedding.

Time	Activity
0:35–1:05	**Attention maps.** Run the pretrained ViT, extract attention from the class token in the final block, reshape to 14×14 and overlay on the photograph. Repeat for three images, including one where the model is wrong.
1:05–1:25	**Do not over-read it.** Attention is not explanation. Find one image where the attention map looks sensible but the prediction is wrong, and one where attention looks scattered but the prediction is right. Write two sentences on what attention maps can and cannot support.
1:25–1:45	**The scaling term.** Remove $\sqrt{d_k}$ from the attention computation and inspect the softmax output distribution. Show it saturating. Explain the gradient consequence in one sentence.
1:45–2:00	**Closing reflection.** Return to Lab 1, where a template classifier failed because someone dimmed the lights. Write 300 words tracing what changed between that failure and this model — and name one thing that has not changed.

Deliverable. The patch grid, three attention overlays including a failure case, the saturation demonstration, and the 300-word reflection. The reflection is assessed.

Expected results. $N = 197$ tokens with the class token; attention maps that often but not always fall on the object; a visibly saturated softmax without the scaling term.

Common pitfalls. Averaging attention across all heads and reporting the mush as "the" attention map; forgetting that the class token occupies index 0 when reshaping to 14×14.

Ethics — the last word. You can now build systems that watch people. Before your demo, state who your system is for, who it is used on, what it does when it is wrong, and who they complain to. A system with no answer to the last question is not finished, whatever its mAP.