FULCRUM COURSE NOTES · 01

Computer Vision: Foundations

How a machine gets from a grid of numbers to something worth calling perception.
Written for people meeting the field for the first time.

FULCRUM

Utsav Poudel
Founder, Fulcrum

ABOUT FULCRUM

Who made this, and why it is free

Fulcrum is a volunteer-run non-profit. We help people from under-resourced regions learn artificial intelligence through one-on-one mentorship, research and publication supervision, project support, workshops, and a guest speaker series.

The premise is simple. Ability is spread evenly across the world; opportunity is not. What separates a capable student in Pokhara or rural Bihar from one at a well-funded lab is rarely intelligence. It is access to supervision, to reviewers, to equipment, and to the unwritten rules nobody writes down.

This deck is part of how we try to close that. It is free, it will stay free, and you may reuse and adapt it for your own teaching under CC BY 4.0. If it is useful to you, come and find us.

Find us

Website
fulcrumgo.github.io

More material
fulcrumgo.github.io/resources

Discord
discord.gg/gbQCGkupdJ

LinkedIn
linkedin.com/company/gofulcrum

Every deck we publish is free to download at fulcrumgo.github.io/resources

PART ONE

An image is a grid of numbers. That is the whole starting point.

Every technique in this deck is a way of answering one question: which arrangements of those numbers mean something?

What a computer actually receives

There is no picture. There is an array.

- A greyscale image is a 2-D array of intensities, usually `0 to 255` per pixel.
- A colour image is three stacked arrays (red, green, blue) so shape is `height × width × 3`.
- **Resolution** is how many samples you took of the world; **bit depth** is how finely you recorded each one.
- Everything downstream is arithmetic on this array. Nothing more mysterious than that.

Worth sitting with: a face and a picture of static are the same kind of object to the machine. Only the pattern differs.

Try this first

Load an image with NumPy, print `.shape`, then set the green channel to zero and look at the result. Ten minutes of that teaches more than an hour of slides.

Colour spaces, and why RGB is often the wrong one

- **RGB** matches how screens emit light, not how people describe colour.
- **HSV** splits hue from saturation and brightness, far easier for 'find the red things' under changing light.
- **Greyscale** discards colour entirely. Often the right call: it cuts data by two-thirds and many tasks do not need it.
- **LAB** separates lightness from colour in a way closer to human perception, which matters for measuring colour difference.

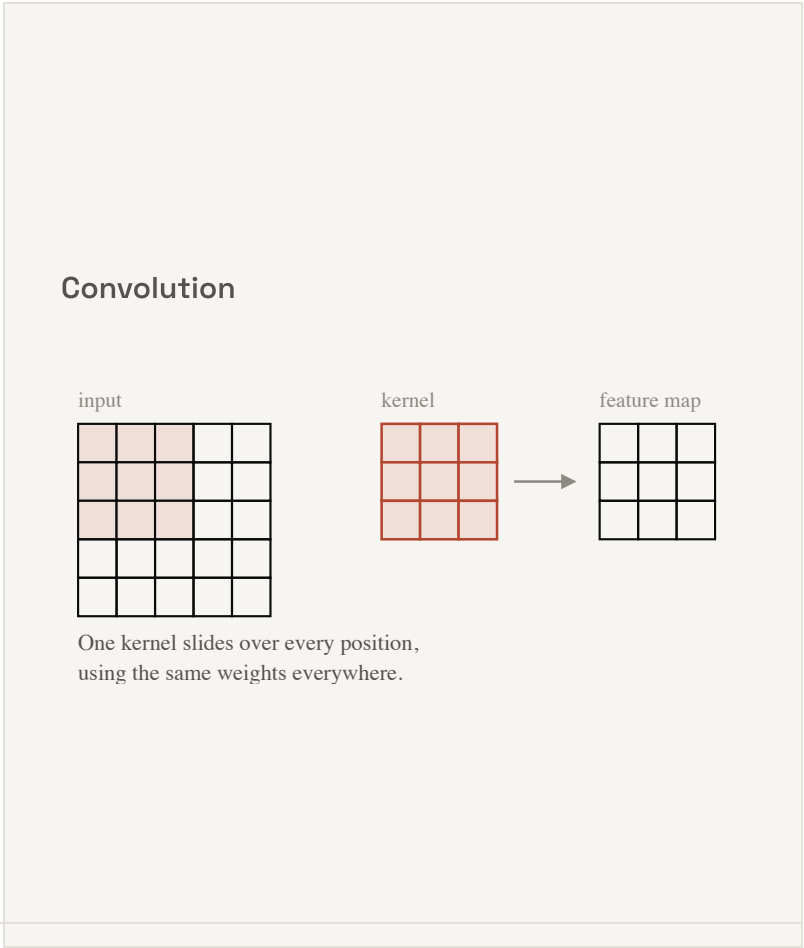
Rule of thumb: if lighting varies and you care about the object's own colour, leave RGB.

Filtering: the one operation to understand properly

A kernel is a small grid of weights slid across the image, multiplying and summing as it goes.

- **Blur** (averaging or Gaussian) suppresses noise and fine detail.
- **Sharpen** amplifies the difference between a pixel and its neighbours.
- **Edge kernels** (Sobel, Prewitt) respond where intensity changes quickly.
- Change only the numbers in the kernel and you change what the filter detects. The machinery stays identical.

This is the exact operation a convolutional network performs. The difference is that a CNN learns the kernel weights instead of you choosing them.



Edges, and what they are really telling you

- An edge is a **steep gradient** in intensity, the image's rate of change, not its value.
- **Sobel** estimates that gradient in x and y; magnitude gives edge strength, direction gives orientation.
- **Canny** adds the parts that make it usable: noise smoothing, thinning edges to one pixel, and hysteresis thresholding so weak edges survive only when connected to strong ones.
- Edges are cheap, fast, and surprisingly robust to lighting, which is why they held the field for decades.

They also fail honestly: a shadow boundary and an object boundary look identical to a gradient operator. That failure is what pushed the field toward learned features.

Thresholding and segmentation

Splitting an image into regions that belong together.

- **Global thresholding** picks one cut-off for the whole image. Fine for a scan, hopeless under uneven light.
- **Otsu's method** chooses that cut-off automatically by maximising separation between the two resulting groups.
- **Adaptive thresholding** computes a local cut-off per neighbourhood, the fix for uneven illumination.
- **Morphological operations** (erode, dilate, open, close) clean up the ragged result afterwards.

Classical features: SIFT and friends

Before learned features, the task was to hand-design descriptors that survive change.

- A good feature is **repeatable** (found again in another view) and **distinctive** (not confusable with everything else).
- **SIFT** (Lowe, 2004) finds keypoints across scales and describes each by local gradient orientations, robust to scale, rotation and moderate lighting change.
- **ORB** is a fast, free alternative built from FAST keypoints and BRIEF descriptors.
- Matching descriptors between two images is what powers panorama stitching, visual odometry, and object instance recognition.

These are not obsolete. For geometry problems (stitching, structure-from-motion, calibration), classical features are still often the right tool.

PART TWO

Where hand-designed features run out

You can design a descriptor for corners. You cannot realistically design one for 'cat'.

The learned-feature turn

- Hand-designed features encode what **we** think matters. That works for geometry and breaks for semantics.
- A convolutional network learns its kernels from data, so the features are whatever reduces the loss.
- **AlexNet** (2012) cut ImageNet error dramatically and made the argument impossible to ignore.
- Three things had to arrive together: large labelled datasets, GPUs, and a few training tricks that stopped deep networks diverging.

The honest version

The ideas were largely in place by the late 1990s. What changed was data and compute. Remember that when you read that some result is a conceptual breakthrough.

How a CNN sees, layer by layer

- **Early layers** respond to edges and colour blobs, strikingly similar to the filters we used to design by hand.
- **Middle layers** combine those into textures, corners, and repeated motifs.
- **Late layers** respond to object parts and whole objects.
- **Pooling** and strided convolution shrink the spatial grid, so each later neuron sees a larger region of the original image, its *receptive field*.

Nobody specified this hierarchy. It falls out of training, and it looks broadly the same across architectures and datasets.

The three tasks you will meet first

- **Classification**, one label for the whole image. 'This is a chest X-ray showing cardiomegaly.'
- **Detection**, labels plus boxes. 'There are three people, here, here and here.'
- **Segmentation**, a label for every pixel. Semantic segmentation labels classes; instance segmentation separates individuals of the same class.
- Difficulty and annotation cost rise sharply across those three. Choose the loosest one that solves your problem.

A lot of wasted effort comes from reaching for segmentation when classification would have answered the question.

Evaluating honestly

The most common failure in student vision projects is not the model. It is the evaluation.

- **Accuracy lies on imbalanced data.** If 97% of scans are normal, predicting 'normal' always scores 97%.
- Use **precision** and **recall**, and be explicit about which error is worse in your setting.
- For detection, **IoU** measures box overlap; **mAP** averages precision across recall levels and classes.
- Split your data **before** you look at it, and never tune against the test set.

If your accuracy looks remarkable, assume leakage until you have proved otherwise. Near-duplicate images across splits is the usual culprit.

Data: the part nobody puts in the paper

- **Augmentation** (flips, crops, colour jitter, rotation), buys generalisation cheaply. Make sure the transform preserves the label.
- **Normalisation** to a consistent mean and scale makes optimisation far better behaved.
- **Class imbalance** can be handled by resampling or by weighting the loss; both are better than ignoring it.
- **Label noise** is normal, not exceptional. Inspect a random hundred examples by hand before you trust any dataset.

Time spent looking at your data returns more than time spent trying architectures. This is consistently true and consistently ignored.

Getting started without a GPU

- **Transfer learning**, take a pretrained backbone, replace the final layer, fine-tune. Works with hundreds of images rather than millions.
- Free hosted notebooks (Colab, Kaggle) give usable GPU time at no cost.
- Start with small images. `128×128` trains fast and tells you whether your pipeline is correct.
- Get an intentionally overfitted model working on ten examples first. If it cannot memorise ten, the bug is in your code, not your model.

Where compute stops being the blocker

For most learning projects it never was. Understanding of the problem, and clean data, run out long before GPU hours do.

ABOUT FULCRUM

Next: Computer Vision, Advanced

Detection and segmentation architectures, vision transformers, generative models, and video.
Bring a project you actually want to build.

Fulcrum is a volunteer-run non-profit helping people from under-resourced regions learn AI through free mentorship, research supervision, and open material like this deck. Founded 2025 in Kathmandu, Nepal. Nothing we do costs anything.

Website
fulcrumgo.github.io

Every deck, free
fulcrumgo.github.io/resources

Apply for mentorship
discord.gg/gbQCGkupdJ

Partnerships
linkedin.com/company/gofulcrum