African Institute for Mathematical Sciences — Cameroon
Structured Master's, Data Science Stream · Computer Vision

What If the Filters Learned Themselves?

Week 1 · Day 5: From Engineered to Learned Features

Yusuf Brima, Ph.D.

Introduction to Computer Vision · AIMS Cameroon, 2026–2027

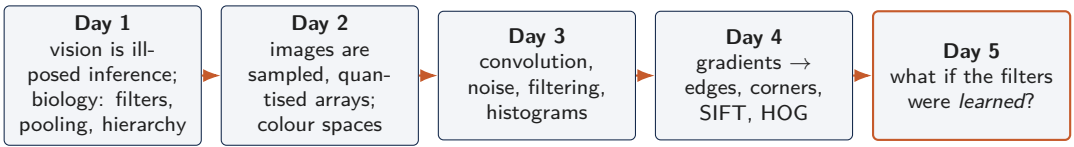
Today

1. **Recognition as learning**: features, classifiers, sliding windows
2. **Three landmark classical systems**: Viola–Jones, HOG + SVM, bag of visual words
3. **An experiment**: does the representation matter?
4. **Learning a filter** by gradient descent
5. **From a neuron to a convolutional layer**: the bridge to Week 2

By the end of today you should be able to

- describe the "features + classifier" pipeline and its limits;
- explain how Viola–Jones, HOG + SVM and bag of words work;
- derive the gradient for learning a filter;
- count the parameters of dense and convolutional layers;
- say what CNNs keep from classical vision, and what they change.

Week 1 so far



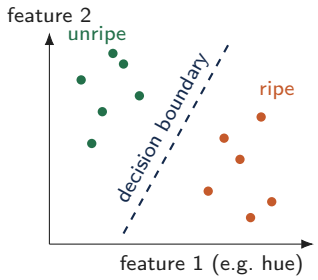
Intuition

Every day this week a *human* chose the kernel: Gaussian, Sobel, Laplacian, the HOG cell size. Today we ask what happens if we let *data* choose. This one question is the doorway from classical to modern computer vision.

Part 1

Recognition as learning

What a classifier does



- Each image becomes a point in a **feature space**.
- A classifier draws a **boundary** between classes, learned from labelled examples.
- A new image is classified by which side it falls on.

A linear classifier

$$\text{score} = \mathbf{w}^\top \phi(\mathbf{x}) + b, \qquad \hat{y} = \begin{cases} \text{ripe} & \text{score} > 0 \\ \text{unripe} & \text{otherwise} \end{cases}$$

Intuition: if the features separate the classes, *any* reasonable classifier works; if they are tangled, no classifier can fix it.

Worked example: ripe or unripe banana?

One hand-designed feature: mean hue

From Day 2: hue is robust to lighting. Ripe bananas are yellow (hue $\approx 55°$), unripe ones green ($\approx 90°$).

Classifier: score $= -0.1 \cdot \text{hue} + 7$.

Ripe banana, hue $55°$: score $= -5.5 + 7 = 1.5 > 0$: **ripe**.
Unripe banana, hue $90°$: score $= -9 + 7 = -2 < 0$: **unripe**.

The boundary sits at hue $= 70°$.

Where it breaks

- Overripe bananas with brown spots: mean hue shifts.
- A green leaf behind the banana: the mean includes background.
- Plantain vs. sweet banana: hue says nothing about *type*.

Intuition

Each failure needs a *new feature*: spot texture, a segmentation step, shape. Feature design never ends.

The classical recognition pipeline



The classical recipe

$$\hat{y} = g_\theta\big(\phi(\mathbf{x})\big), \qquad \theta^\star = \arg\min_\theta \sum_i \ell\big(g_\theta(\phi(\mathbf{x}_i)),\, y_i\big)$$

Only θ is learned; ϕ is fixed by a human.

- ϕ: HOG, SIFT histograms, colour histograms, Haar features, texture descriptors.
- g_θ: linear SVM, logistic regression, random forest, boosting.
- The classifier is the easy part. Success lives or dies in ϕ.

From scores to probabilities: softmax

- With K classes, a linear classifier gives one score per class: $s_k = \mathbf{w}_k^\top \phi(\mathbf{x}) + b_k$.
- Softmax turns scores into probabilities that are positive and add up to 1.

Softmax

$$p_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

This is *logistic regression* for K classes, and the last layer of every classification CNN (Day 1's ResNet-50 had $K = 1000$).

Three classes

Scores: cat 2.0, dog 1.0, weasel 0.1.

$e^{2.0} = 7.39, \quad e^{1.0} = 2.72, \quad e^{0.1} = 1.11;$ sum $= 11.22$.

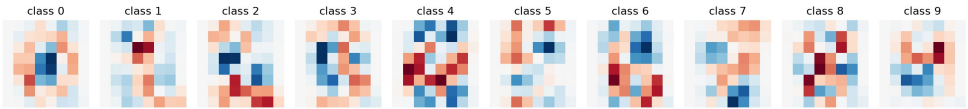
$p = (0.66,\ 0.24,\ 0.10)$.

Add 1 to every score: the probabilities do not change. Only *differences* between scores matter.

Intuition

Training pushes the correct class's score up and the others down, using exactly the gradient-descent rule we meet later today.

A linear classifier on pixels is a template



The learned weights of logistic regression on 8×8 digit images, one image per class (red: positive, blue: negative). The "0" weights are a ring with a negative centre; "1" is a vertical bar.

Intuition

Each class score is a dot product of the image with a learned template: Day 4's template matching. That is why it fails under shifts: one template per class, at one position.

Training, testing and generalisation

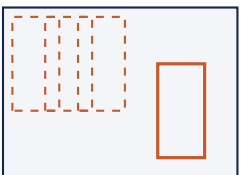
- **Training set**: images with labels; the classifier adjusts θ to get them right.
- **Test set**: images it has *never seen*. Only accuracy here tells you how it will do in the world.
- **Overfitting**: memorising the training set (including its noise) instead of learning the pattern.

Leakage traps in vision

- Near-duplicate photos (burst shots) split across train and test.
- Several slides from the *same patient* in both sets.
- Same field, same day, same phone in train and test: the model learns the conditions, not the disease.

Split by patient, field or day, not by image.

From classification to detection: sliding windows



slide a fixed window; classify each one: "person" or "not person"

1. Build an image pyramid (Day 4) so objects of every size fit the window at some level.
2. At every level, slide the window with a small stride.
3. Compute ϕ and the classifier score for each window.
4. Keep high-scoring windows; merge overlapping ones.

How many windows?

A 1280×960 photo, 64×128 window, stride 8: $153 \times 105 \approx 16\,000$ windows at one scale. A pyramid with scale step 1.05 brings it to about **170 000**.

Intuition: detection becomes classification, many thousands of times. The classifier must be fast and almost never fire on background.

Merging overlapping detections: IoU and non-maximum suppression

Intersection over union

$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \in [0, 1]$$

Two boxes

A: corners $(0, 0)$–$(10, 10)$; B: $(5, 5)$–$(15, 15)$.
Intersection: $5 \times 5 = 25$. Union: $100 + 100 - 25 = 175$.
IoU $= 25/175 \approx 0.14$: barely overlapping.

Non-maximum suppression for boxes

1. Sort boxes by score.
2. Keep the best; delete every other box with IoU > 0.5 with it.
3. Repeat with the remaining boxes.

Intuition

Same idea as Canny's NMS (Day 4): keep the local maximum, suppress its neighbours. YOLO and every modern detector still do this.

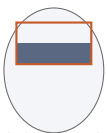
Part 2

Three landmark classical systems

Viola–Jones (2001): faces in real time



Haar-like features: sum(white) − sum(dark)



eyes darker than cheeks:
a "two-rectangle" feature fires

- **Features**: differences of rectangle sums, at many sizes and positions: about 160 000 per 24×24 window.
- **Integral image**: any rectangle sum in 4 lookups.
- **AdaBoost** picks the few hundred most useful features.
- **Cascade**: cheap early stages reject most windows immediately.

Intuition

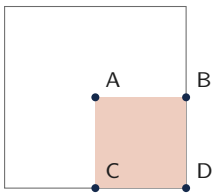
Faces have a consistent *pattern of light and dark*: darker eyes, a bright nose bridge, a darker mouth. Rectangle differences capture exactly that, very cheaply.

Worked example: the integral image

Integral image

$$S(x,y) = \sum_{x' \le x,\ y' \le y} I(x',y')$$

Any rectangle sum $= S_D - S_B - S_C + S_A$ (its four corners).



A 3×3 image

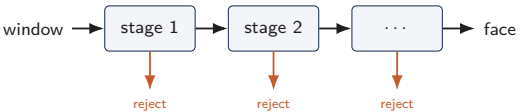
$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 3 & 6 \\ 5 & 12 & 21 \\ 12 & 27 & 45 \end{bmatrix}$$

Sum of the bottom-right 2×2 block $(5+6+8+9 = 28)$:
$S_D - S_B - S_C + S_A = 45 - 6 - 12 + 1 = 28$. ✓

Intuition

One pass over the image, then every rectangle, of any size, costs 4 lookups. Speed comes from a clever *representation*, again.

The cascade: reject the easy windows fast



- Stage 1 uses only 2 features and throws away about half of all background windows.
- Later stages are more expensive but see very few windows.
- On average, a background window costs only a handful of features.

Ten stages

Each stage keeps 99.7% of faces and lets through 30% of non-faces.
Faces found: $0.997^{10} \approx 97\%$. False alarms: $0.3^{10} \approx 6 \times 10^{-6}$, about one per 170 000 windows: one per photo.

Viola–Jones on a real photograph



Photokadaffi, CC BY-SA 4.0

OpenCV's pre-trained frontal-face cascade on the Mbouda street scene:

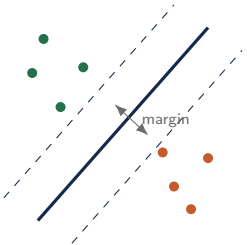
- 3 faces found: the ones looking roughly at the camera.
- Faces in profile, turned away or small are missed.
- 1 false alarm on the cargo sack.

Intuition

The cascade was trained mostly on frontal faces in 2001 datasets. Its features encode "frontal face", nothing more.

HOG + linear SVM (2005): people detection

- Slide a 64×128 window over an image pyramid.
- Describe each window with HOG: 3780 numbers (Day 4).
- A **linear SVM** scores "person" vs. "not person".
- Merge overlapping detections with NMS.



Intuition

An SVM draws the boundary with the **widest margin** to the nearest examples of each class: the safest line, the one least likely to be fooled by small changes.

HOG + SVM on a real street



Bdx, CC BY-SA 4.0

OpenCV's pre-trained HOG people detector on the Yaoundé street:

- finds the woman in red (twice: two overlapping boxes that NMS should merge);
- **misses** the man in front, cut off by the image border and seen from behind, and the people further away.

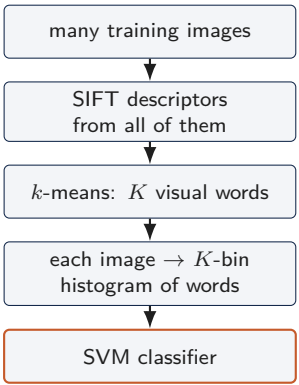
On the *same photo*, the YOLOv12 network from Day 1 found 6 people.

Intuition

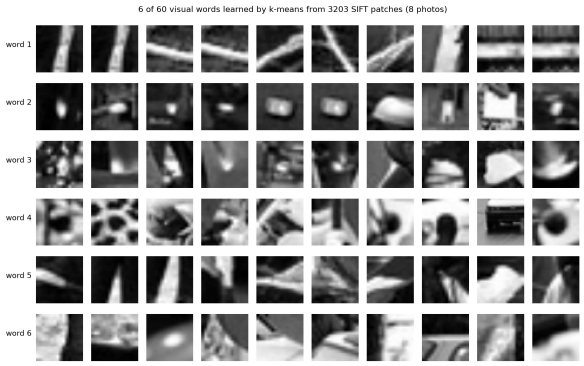
A fixed 64×128 upright template cannot cover the variety of real poses, sizes and occlusions.

Bag of visual words: images as documents

- In text classification, a document is summarised by *word counts*, ignoring word order.
- Do the same for images: what are the "words"?
- **Answer**: cluster many SIFT descriptors with k-means. Each cluster centre is a **visual word**: a typical local pattern.



Visual words from real photographs



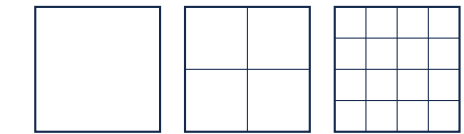
We extracted 3203 SIFT patches from 8 of this week's photos and clustered them into 60 words. Each row shows the patches closest to one word.

- Some words are clean: thin bright bars, bright spots, dark corners.
- Others mix patterns that only *look* alike to SIFT.

One image as a histogram

A street photo might contain word 1 (bars) 40 times, word 2 (spots) 12 times, ...: a 60-number vector, whatever the photo's size.

Adding back position: spatial pyramids



- Compute word histograms for the whole image, its 4 quarters and its 16 sixteenths, and concatenate them.
- Now the description knows that "sky words" are at the top and "road words" at the bottom.
- Lazebnik et al. (2006): a big accuracy gain for scene classification.

Length of the descriptor

With $K = 200$ visual words: $(1 + 4 + 16) \times 200 = 4200$ numbers per image.

Intuition: coarse position at several scales, like HOG's grid of cells and a CNN's shrinking feature maps.

Bag of words: strengths and blind spots

Strengths

- Fixed-length vector for any image size.
- Inherits SIFT's invariance to scale, rotation, lighting.
- Robust to occlusion: a hidden part removes a few words, not the whole description.
- With Fisher vectors, the best ImageNet entries of 2010–2011.

Blind spots

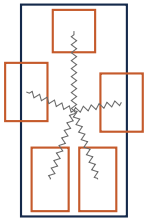
- Ignores *where* words are: a face with eyes below the mouth looks the same. Spatial pyramids add coarse position.
- Words are learned without labels: k-means does not know which patterns matter for the task.

Intuition

Notice the theme: HOG, SIFT and bag of words are all **histograms of local responses**. Local filters, pooled into histograms, then normalised. A CNN is the same recipe, but every stage is learned.

Deformable part models (2008–2010): the last great classical detector

- One rigid HOG template cannot cover bent arms and turned heads.
- **DPM**: a coarse *root* template for the whole person, plus several finer *part* templates (head, arms, legs).
- Parts may move relative to the root, at a cost, like springs.
- Score = root match + part matches − spring penalties.



root + parts on springs

Intuition

DPM was a hand-designed *hierarchy*: small parts combined into a whole. Deep networks learn many such levels automatically, which is why they overtook DPM within two years of AlexNet.

What hand-engineering bought, and where it ran out

Strengths

- Interpretable: you know what each number means.
- Data-efficient: invariances are *built in*, not learned from examples.
- Cheap: runs on a CPU or a microcontroller.
- Still the right tool for matching and geometry, and for tiny-data problems.

Limits

- Every new task needs a new expert-designed ϕ.
- Low- and mid-level only: no hand-made feature says "wheel", "lesion" or "parasite".
- A fixed ϕ caps accuracy: more data stops helping.
- ImageNet 2011 best: 25.8% top-5 error. AlexNet 2012: **15.3%**.

The deep-learning idea, in one line

$$\hat{y} = g_\theta\big(\phi_w(\mathbf{x})\big), \qquad (\theta^\star, w^\star) = \arg\min \sum_i \ell(\cdot) \quad : \text{learn the features too.}$$

Part 3

An experiment: does the representation matter?

The set-up: handwritten digits

- scikit-learn digits: 1797 small images of handwritten 0–9, upsampled to 24×24.
- 70% for training, 30% for testing.
- **Same classifier** (standardise + logistic regression) on two representations:
  1. the 576 raw pixel values;
  2. HOG features (Day 4).
- Then test on digits **shifted** by 1, 2 and 3 pixels.

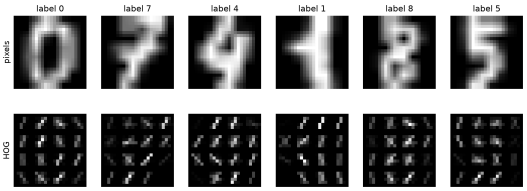
Intuition

This is a controlled experiment: only ϕ changes. Any difference in accuracy is due to the representation alone.

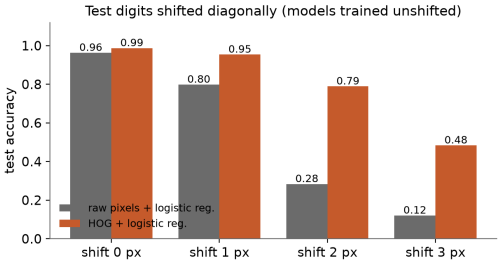
Practitioner's Note

A shift of 2 pixels on a 24×24 digit is like your subject moving a few centimetres in a photo: a change no human would even notice.

Results: pixels vs. HOG



Test shift	Raw pixels	HOG
0 px	0.963	0.987
1 px	0.798	0.954
2 px	0.283	0.789
3 px	0.120	0.483



Intuition

Without shifts, both are excellent. Shift by two pixels and raw pixels collapse to 28% (barely above the 10% of guessing); HOG keeps 79%.

Why does HOG degrade gracefully?

Following one stroke

A vertical stroke moves 2 pixels right.

Raw pixels: the stroke now occupies *different* input dimensions. To a linear classifier, these are completely new numbers.

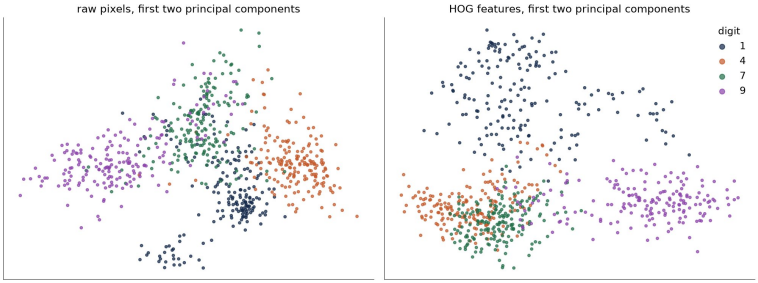
HOG: the stroke stays inside the same 6×6 cell (mostly). Its cell histogram still says "one vertical edge". The feature vector barely changes.

- The step responsible is **pooling** gradient votes into cell histograms (Day 4).
- At 3 pixels many strokes cross into the neighbouring cell, and HOG starts to suffer too.
- Invariance is *built in* up to the cell size, and no further.

Practitioner's Note

CNNs get the same effect from pooling and strides, and they also learn it from *data augmentation*: show shifted copies during training.

A picture of the representation



Four digit classes projected to their two main directions of variation (PCA). In raw pixels the classes overlap broadly. In HOG space the 1s and 9s form clean separate groups; 4s and 7s still overlap in these two dimensions.

Intuition

A good representation pulls same-class examples together and pushes classes apart. Learning a representation means learning a space where this is true *for your task*.

Teaching invariance with examples: data augmentation



RyanDannyChacko, CC BY-SA 4.0

HOG builds invariance into the *features*. A learned model can instead be *shown* the variations: every training image is randomly flipped, cropped, rotated, re-lit, blurred or partly hidden. The label stays "cat".

Intuition

Each augmentation is one of Day 1's challenges: pose, scale, illumination, noise, occlusion. Augmentation says: "these changes do not matter; learn to ignore them".

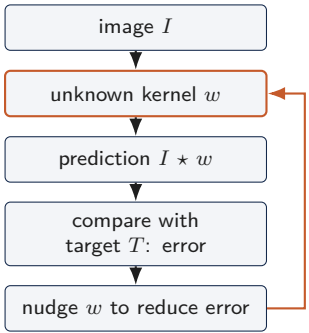
Part 4

Learning a filter by gradient descent

Can data rediscover Sobel?

The set-up

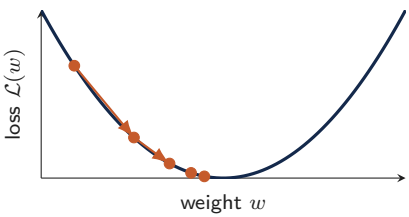
- We have an image I and a target edge map $T = I \star G_x$, but pretend we do not know G_x.
- Start with a **random** 3×3 kernel w.
- Adjust w, a little at a time, so that $I \star w$ looks more and more like T.



Intuition

This is the whole of deep learning in miniature: a model with adjustable numbers, a measure of how wrong it is, and a rule for improving the numbers.

Gradient descent, intuitively



- The loss measures how wrong the model is for each choice of weights.
- The gradient points *uphill*; step the other way.
- Near the bottom the slope flattens, so steps shrink automatically.

One step

$\mathcal{L}(w) = w^2$, slope $2w$. At $w = -1.8$ with $\eta = 0.2$:
$w \leftarrow -1.8 - 0.2 \cdot (-3.6) = -1.08$.

Practitioner's note: learning rate too small: painfully slow; too large: overshoots and diverges. Choosing η is the first knob you will turn in Week 2.

The loss and its gradient

Mean squared error

$$\mathcal{L}(w) = \frac{1}{N} \sum_{p} \Big(\underbrace{\langle w,\, P_p \rangle}_{(I \star w)[p]} - T[p] \Big)^2$$

P_p: the 3×3 patch around pixel p. $\langle w, P_p \rangle$: the filter output there (a dot product).

Gradient descent

$$w \leftarrow w - \eta \, \frac{\partial \mathcal{L}}{\partial w}$$

η: learning rate (step size).

Deriving the gradient

Write the error at p as $e_p = \langle w, P_p \rangle - T[p]$.

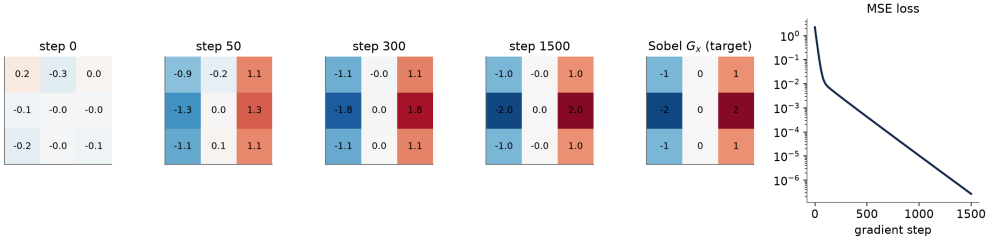
Only e_p depends on w_{ij}, linearly: $\partial e_p / \partial w_{ij} = P_p[i, j]$.

Chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{2}{N} \sum_{p} e_p \, P_p[i, j]$$

In words: correlate the *error map* with the image.

Yes: data rediscovers Sobel



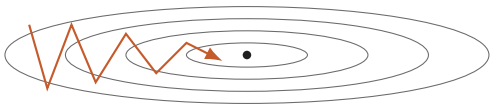
- From a random start, the kernel converges to $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, using only input/output examples.
- No human wrote the numbers $-1, -2, -1$ and $1, 2, 1$.

Intuition

The loss is a bowl (it is linear least squares): one global minimum, and gradient descent rolls into it. Deep networks are not bowls, but the same rule still works remarkably well.

Why did it take 1500 steps?

- In natural images, neighbouring pixels are strongly *correlated*: the nine values in a patch move together.
- So the bowl is long and narrow (ill-conditioned; condition number $\kappa \approx 570$). Gradient descent zig-zags across the valley.
- On white-noise images (uncorrelated pixels), the bowl is round and it converges in fewer than 10 steps.



narrow valley: gradient descent zig-zags

Practitioner's Note

This is why we *normalise inputs* (Day 2), use *batch normalisation* and *adaptive optimisers* (Adam) in Week 2: all make the bowl rounder.

From “match Sobel” to “match the label”

- Today’s target was a *known* edge map, so we recovered a known filter.
- Now change the target to a **class label**: “malaria parasite” or “healthy cell”.
- Stack many filters, with non-linearities in between, and train all of them by the same rule.
- The filters become **whatever the task needs**, including patterns no human would have thought to design.

Intuition

Hand-engineering asks: “what should the computer look for?” Learning asks: “what examples can I show it?”. The second question is often easier to answer, if you have the examples.

Practitioner’s Note

If a human labels the edges instead, the target is noisy and subjective: the learned filter becomes the *average* labeller’s notion of an edge. Label quality sets the ceiling.

Part 5

From one filter to a convolutional layer

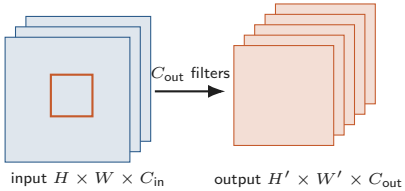
A neuron is a filter at one location

One artificial neuron

$$z = \sigma\Big(\sum_{i,j,c} w_{ijc}\, x_{i,j,c} + b\Big)$$

x: a patch with c colour channels; w: the kernel; b: a bias; σ: a non-linearity.

- The sum is exactly one output pixel of a convolution (Day 3).
- Slide the *same* neuron over the image: a **feature map**.
- Learn C_{out} different kernels at once: a **convolutional layer**.



Parameters of a conv layer

$$(K \cdot K \cdot C_{\text{in}} + 1) \cdot C_{\text{out}}$$

Why the non-linearity matters

ReLU

$$\sigma(u) = \max(0, u)$$

Keep positive responses, zero out negative ones.

Two layers without it

$(I * w_1) * w_2 = I * (w_1 * w_2)$ by associativity (Day 3).

Two linear layers are just *one* bigger linear filter. A hundred layers would be one filter too.

- ReLU breaks the chain: layer 2 now works on "where layer 1 fired", not on a blend of pixels.
- That lets the network build **parts from edges** and **objects from parts**.
- Biologically: a neuron cannot fire at a negative rate.

Intuition

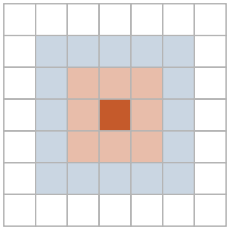
ReLU is like Canny's threshold or the rectification in HOG's magnitudes: a decision "is this pattern here or not?" between stages.

Pooling and growing receptive fields

- **Max-pooling**: keep the largest value in each 2×2 block. Halves the size; tolerates small shifts. Like complex cells in V1 (Day 1) and HOG's cells (Day 4).
- **Receptive field**: the region of the input that affects one output.

Stacked 3×3 convolutions

One layer sees 3×3; two see 5×5; L layers see $(2L+1) \times (2L+1)$.
Ten layers: 21×21. Add a $2\times$ pooling every two layers and the receptive field roughly doubles each time.



1 pixel $\rightarrow 3 \times 3 \rightarrow 5 \times 5$

Intuition

Early layers see small details; deep layers see the whole object. The ventral stream again (Day 1).

Why convolution and not a dense layer?

Dense layer on a $224 \times 224 \times 3$ image

$150\,528$ inputs $\times$ 1000 units $\approx$ **150 million** weights, for *one* layer, with no notion of locality and no reuse across positions.

Conv layer: 3×3, 3 $\to$ 64 channels

$(3 \cdot 3 \cdot 3 + 1) \cdot 64 = \mathbf{1\,792}$ parameters, used at every position.

Three priors built into convolution

1. **Locality**: small receptive fields (retina, V1).
2. **Weight sharing**: the same detector everywhere (Day 3).
3. **Hierarchy**: stack layers; receptive fields grow (V1 $\to$ IT).

Intuition

These priors are knowledge about images, built into the architecture so it does not have to be learned from data.

Worked example: one big filter or two small ones?

64 → 64 channels

One 5×5 layer: $(5 \cdot 5 \cdot 64 + 1) \cdot 64 = 102\,464$ parameters. Receptive field 5×5.

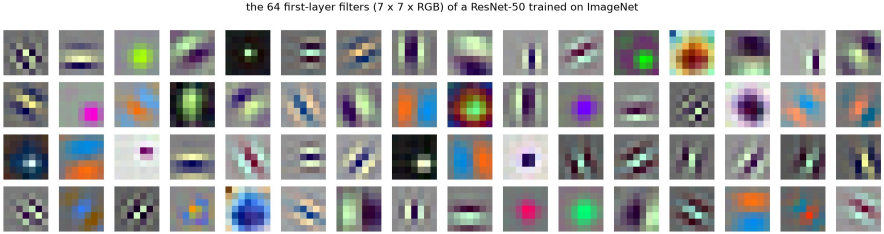
Two stacked 3×3 layers: $2 \times (3 \cdot 3 \cdot 64 + 1) \cdot 64 = 73\,856$ parameters. Receptive field also 5×5.

- Same view of the image, **28% fewer parameters**.
- **Two** ReLUs instead of one: more non-linear, so more expressive.
- This is the key design idea of VGG (2014), which you will meet on Monday.

Practitioner's Note

Counting parameters by hand is a skill you will use every day in Week 2: to estimate memory, to spot overfitting risk, and to check that a model you build matches the paper.

What learned filters actually look like

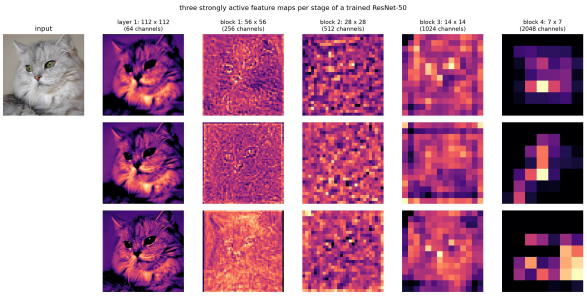


The real first-layer weights of the ResNet-50 we used on Day 1, learned from 1.28 million ImageNet photos. Nobody designed them.

Intuition

Look: oriented edge detectors at many angles (Sobel, Gabor, V1 simple cells), centre–surround blobs (the retina's DoG), and colour-opponent filters. Data rediscovered this week's toolbox.

Deeper layers: from edges to parts



Gourdo 1 at en.wikipedia, Public domain

The first layer's maps still look like the cat: edges and shading. Deeper maps get smaller and more abstract. By block 4 (7×7) each map says roughly "*this kind of part* is *here*", for 2048 kinds of part.

Intuition

Exactly the hierarchy of Day 1: retina $\rightarrow$ V1 (edges) $\rightarrow$ V4 (shapes) $\rightarrow$ IT (objects). Receptive fields grow; features become more abstract and more invariant.

Engineered vs. learned: side by side

	Classical (Week 1)	Deep (Week 2)
Filters	designed: Gaussian, Sobel, Gabor	learned from data
Non-linearity	thresholds, magnitude, max	ReLU
Pooling	HOG cells, SIFT histograms	max / average pooling, strides
Normalisation	HOG blocks, SIFT unit length	batch / layer normalisation
Scale	image pyramids, scale space	feature pyramids inside the network
Classifier	SVM, logistic regression	final linear layer + softmax
Training data	hundreds to thousands	thousands to millions

Intuition

A CNN **keeps** the structure of classical vision (filter, rectify, pool, normalise, repeat) and **changes** who chooses the numbers.

Learned features can learn the wrong thing

- A learned model uses *whatever* pattern predicts the label in the training data, wanted or not.
- Documented example: pneumonia classifiers trained on chest X-rays from several hospitals partly learned to recognise *which hospital* an image came from (scanner type, text markers), because hospitals differed in how sick their patients were (Zech et al., 2018).
- Day 1's weasel is a milder case: rock texture plus a small head.

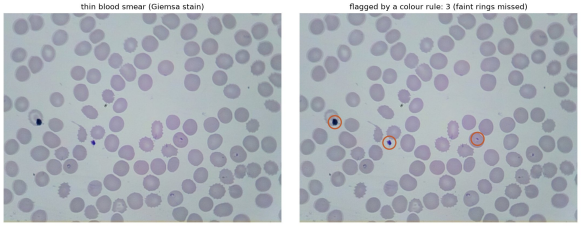
Intuition

Hand-designed features can only use what the designer put in. Learned features can use anything, including shortcuts. That is their power *and* their risk.

Practitioner's Note

Defences: test on data from a *different* site or device; inspect what the model looks at (the occlusion map from Day 1); balance confounders in your training set.

Case study: malaria microscopy, the classical way



OpenFlexure Project, CC BY-SA 4.0

The pipeline we built this week

1. Convert to HSV (Day 2).
2. Threshold dark, saturated purple (Day 3).
3. Clean the mask with morphology (Day 3).
4. Count connected components (Day 2).

- No training data needed; every step explainable.
- Found 3 parasites; missed the faint ring forms.

Case study: malaria microscopy, the learned way

What a learned detector changes

- A CNN trained on thousands of expert-annotated smears learns what a ring form looks like, including faint ones.
- It can learn to ignore stain variation and artefacts, *if* the training data contain them.
- It needs labelled data, compute, and validation on *local* slides and microscopes.

Intuition

Neither approach is simply "right". Strong systems often combine them: classical colour normalisation and cell segmentation feeding a learned classifier.

Practitioner's Note

Before trusting either, ask the Day 1 question: was it tested on slides like the ones it will see, from the same kind of microscope, stain and patients?

When should you still use classical methods?

Classical is often the right choice when

- you have very little labelled data;
- the setting is controlled (factory line, scanner, microscope with fixed stain);
- you need geometry: matching, stitching, measuring, 3D;
- you must run on a microcontroller or offline phone;
- you must explain every decision.

Deep learning wins when

- the categories are varied and natural (crops, animals, people, lesions);
- you can collect or borrow lots of data (or a pre-trained model);
- accuracy matters more than transparency.

Practitioner's Note

In practice, systems mix both: classical preprocessing (CLAHE, colour normalisation, tiling) feeding a learned model, and classical geometry checking its outputs.

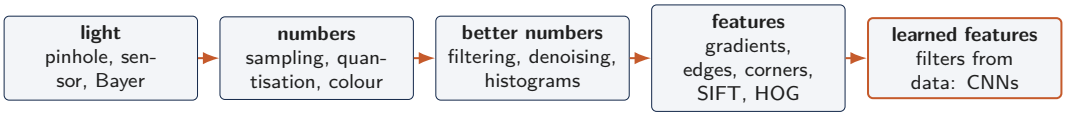
Part 6

Wrapping up Week 1

Key takeaways

1. Classical recognition = designed features ϕ + learned classifier. **Success lives in ϕ.**
2. Viola–Jones, HOG + SVM and bag of words: cheap, clever, but limited to what their designers anticipated.
3. Representation matters: same classifier, 28% vs. 79% accuracy under a 2-pixel shift.
4. A filter can be **learned** by gradient descent; data rediscovered Sobel.
5. A neuron = a filter at one location; a conv layer = many shared filters.
6. CNNs keep classical structure (filter, rectify, pool, normalise) and learn the numbers.

Week 1 in one picture



What carries into Week 2

- Output-size formula $\lfloor (N + 2p - K)/s \rfloor + 1$
- Normalisation, `dtype` and layout discipline
- Data augmentation = Day 1's nuisance variations
- IoU and NMS for detection

Intuition

Week 1 is not "old methods we no longer need". It is the vocabulary in which deep networks are written.

Assessment and preparation for Week 2

This week's assessment

- **Quiz 1**: one hour, covers Days 1–5. Time and room announced by the tutors.
- **Assignment 1**: individual, LaTeX, theory + coding. Due at the start of Week 2, Day 3.

Practise (tutorial notebook `W1D5_Engineered_vs_Learned`)

- Reproduce pixels vs. HOG and the shift experiment.
- Learn Sobel, a Laplacian and a blur by gradient descent.

Before Monday

- Check that PyTorch (≥ 2.0) and torchvision import on your machine, *or* that you can open a Colab GPU runtime.
- Read: course notes Ch. 4; Goodfellow et al., *Deep Learning*, Ch. 9 (convolutional networks).
- Start your list of capstone ideas.

Week 2, Day 1: CNN building blocks and how to train them.

Credits

Photographs from Wikimedia Commons under the licences shown. Other images: scikit-learn `digits`; scikit-image `camera`. SIFT patches for the visual-words figure were taken from the photographs credited across this week's decks. Filters and feature maps: torchvision ResNet-50 (ImageNet weights).

`moto_mbouda`: Photokadaffi, CC BY-SA 4.0
`yaounde_taxis`: Bdx, CC BY-SA 4.0
`cat_persian`: Gourdo 1 at en.wikipedia, Public domain
`douala_deido`: Happiraphael, CC BY-SA 4.0
`douala_ndokoti`: Minette Lontsie, CC BY-SA 4.0
`maize_blight`: Chroanch, CC BY-SA 4.0
`pcb`: Ashley Pomeroy, CC BY 4.0
`leopard_tree`: Davidvraju, CC BY-SA 4.0
`cat_leaves`: RyanDannyChacko, CC BY-SA 4.0
`malaria_smear`: OpenFlexure Project, CC BY-SA 4.0