Images with CNNs

A CNN reads a picture in small windows, keeps what matters, and shrinks it step by

Introduction

A photo is a grid of numbers: millions of pixels, each with red, green and blue values. A plain fully-connected network treats every pixel as an independent input, which is wasteful — a cat's ear looks the same whether it sits in the top-left or the bottom-right of the photo, but a plain network must relearn "ear" at every position separately. It also has far too many weights: a small 200×200 photo already means 40,000 inputs per colour channel.

A **convolutional neural network (CNN)** fixes both problems with one idea: slide a small window, called a **filter** (or kernel), across the whole picture, and at each position ask "how much does this patch look like my pattern?" The same filter is reused everywhere, so the network learns "edge" once and finds edges anywhere. Stack such layers and you get the feature ladder from DL.1: edges → textures → parts → objects.

Two more ideas complete the picture. **Pooling** shrinks each layer's output by keeping only the strongest signal in each small region — the picture gets smaller, the network gets faster, and small shifts of an object stop mattering. And at the end, after the picture has been shrunk to a small set of rich features, ordinary fully-connected layers make the final decision.

In this lesson you will trace a picture through a tiny CNN and learn what each stage sees.

Where beginners get stuck: they picture the filter as something that "looks at" the picture like an eye. It does not look — it multiplies. At each position, the filter's 9 weights multiply the 9 pixel values underneath, and the sum is one number in the feature map. A big number means "this patch matches my pattern well". Slide, multiply, add, move on. Once that mechanical picture clicks, pooling and stacking layers are easy: each new layer just repeats the same multiply-and-add trick on the previous layer's feature maps. *(Roman Urdu: CNN tasveer ko chhoti khirkiyon se parhta hai — aik hi filter poori tasveer par phirta hai, is liye kaun sa hissa kahan hai, is se farq nahi parta.)*

Key points

- **A filter is a small pattern detector.** A 3×3 filter is just 9 weights. Sliding it over the image and multiplying at each position produces a **feature map**: a grid showing where that pattern appears, and how strongly.
- **Weight sharing is the superpower.** The same filter is used at every position, so the network needs far fewer weights than a fully-connected network, and it finds a pattern anywhere in the image (translation invariance, in plain words: position does not matter).
- **Early filters learn simple things, late filters learn complex ones.** First layers find edges and colours; middle layers combine them into textures and parts; late layers respond to whole objects. Nobody programs this ladder — training discovers it.

- **Pooling shrinks and toughens.** Max pooling keeps only the largest value in each small region (for example 2×2 → 1 value). The picture becomes smaller and the network stops caring whether a feature sits one pixel left or right.
- **Typical CNN shape:** several rounds of (convolution → activation → pooling), then the shrunken features are flattened and fed to fully-connected layers, ending in a softmax classifier (DL.3).
- **CNNs also need lots of data — or transfer learning.** Training a CNN from scratch on 200 photos will overfit. The standard shortcut: start from a CNN trained on millions of photos, keep its early layers (they already know edges and textures), and retrain only the last layers for your task.
- **Stride and padding control the feature map's size.** Stride is how many pixels the filter jumps at each step: stride 1 checks every position, stride 2 skips every other one and halves the map. Padding adds a border of zeros around the image so edge pixels get checked as often as centre pixels, and so the map does not shrink too fast.
- **Data augmentation multiplies a small dataset for free.** Flipping, small rotations and crops of a photo are still the same object, so one photo becomes many training examples. Augmentation is the first thing to try before collecting more photos.

Examples

Example 1 — One filter finding vertical edges

Take a 5×5 black-and-white picture of a white square on a black background, and a 3×3 filter that rewards "bright on the right, dark on the left". Slide it over the picture: at the square's left edge the filter fires strongly (dark left, bright right), at the right edge it fires negatively, and in flat areas it stays near zero. The output feature map is bright exactly along the left edge. That is a convolution: one tiny pattern, found everywhere, in one pass.

Example 2 — Tracing a photo through a tiny CNN

Input: a 32×32 photo of a cat.

- **Conv layer 1 (32 filters):** output is 32 feature maps — edges at many angles, colour blobs.
- **Pooling:** each map halves in size (32×32 → 16×16). Small shifts no longer matter.
- **Conv layer 2 (64 filters):** maps now show textures — fur, whisker patterns, eye shapes.
- **Pooling again:** 16×16 → 8×8.
- **Conv layer 3 (128 filters):** maps respond to parts — ears, eyes, paws.
- **Flatten + fully-connected + softmax:** the network weighs all part-signals and outputs "cat 0.91, dog 0.07, bird 0.02".

Example 3 — Transfer learning on a tiny dataset

You have 300 photos of local birds and want an app that names the bird. Training a CNN from scratch would memorise 300 photos and fail. Instead: take a CNN trained on millions of general photos, freeze its early layers, and train only a new final classifier on your 300 photos. The early layers already know feathers, beaks

and edges; your small dataset only has to teach the last step. This is the most practical CNN skill for a student project.

Example 4 — One convolution, worked by hand

A 3×3 patch of a black-and-white picture (1 = bright, 0 = dark) and a 3×3 filter that rewards "bright on the right":

```
Patch:          Filter:
0  0  1         -1  0  1
0  0  1         -1  0  1
0  0  1         -1  0  1
```

Multiply matching positions and add: (−1×0 + 0×0 + 1×1) + (−1×0 + 0×0 + 1×1) + (−1×0 + 0×0 + 1×1) = 1 + 1 + 1 = 3. A strong positive result — this patch is a vertical edge, bright on the right. A patch that is bright on the left would give −3, and a flat patch (all 0s or all 1s) would give 0. That single number, 3, is one cell of the feature map.

Example 5 — Max pooling, worked by hand

A 4×4 feature map, pooled with 2×2 regions (keep the largest value in each region):

```
Input:          Output (2×2):
3  1  0  2       3  2
0  2  1  1       2  4
1  0  4  4
2  2  3  1
```

Top-left region (3, 1, 0, 2) → 3. Top-right (0, 2, 1, 1) → 2. Bottom-left (1, 0, 2, 2) → 2. Bottom-right (4, 4, 3, 1) → 4. The map shrank from 16 numbers to 4, the strongest signals survived, and whether the "4" sat one pixel left or right no longer matters. That is pooling's whole job, in one small grid.

Example 6 — Real world: a plant-disease checker for farmers

A farmer photographs a leaf, and an app names the disease on it — or says the leaf looks healthy. Underneath is a CNN, usually built by transfer learning: a network pre-trained on millions of general photos, with a new classifier head trained on a few thousand labelled leaf photos. Early layers already knew edges and textures; the new head only had to learn leaf spots and colours. The farmer needs no lab test and no expert visit for a first answer, and the app improves as more labelled leaf photos are added. Same CNN ladder as this lesson, applied to a real field problem.

Quick recap

- A CNN slides small filters over the image; each filter makes a feature map.
- The same filter is reused everywhere, so patterns are found at any position with few weights.
- Layers build a ladder: edges → textures → parts → objects.

- Pooling shrinks the maps and makes the network tolerant of small shifts.
- End shape: convolutions + pooling, then fully-connected layers + softmax.
- With little data, use transfer learning instead of training from scratch.

Common mistakes

1. **Feeding a huge photo straight into fully-connected layers** — millions of weights, instant overfitting. Fix: use convolutional layers first; let pooling shrink the picture.
2. **Training a CNN from scratch on a tiny dataset** — it memorises. Fix: transfer learning, or heavy data augmentation (flips, crops, rotations) to multiply your examples.
3. **Forgetting that pooling throws away position** — after heavy pooling the network knows *what* is present but not exactly *where*. Fix: if exact position matters (for example finding a tumour's location), use architectures that preserve location, not plain classification CNNs.

1. **Feeding raw 0–255 pixel values without scaling** — large inputs make training unstable and slow. Fix: scale pixels to 0–1 (divide by 255) before training; it costs one line and helps every CNN.
2. **Pooling away the small thing you need** — heavy pooling shrinks small objects (a tiny spot on a leaf, a distant sign) until they vanish from the feature maps. Fix: pool less aggressively, or use higher-resolution inputs, when small details carry the answer.

Practice

1. A 4×4 picture is max-pooled with 2×2 regions. How big is the output? What information is kept, and what is thrown away?
2. Why does a CNN find a cat's ear anywhere in the photo, while a plain fully-connected network must relearn "ear" at each position? Answer in two sentences.
3. You have 400 photos for a flower classifier. Would you train from scratch or use transfer learning? Give one reason.

1. Work out the output of this 3×3 patch with the Example 4 filter: all rows are "1 0 0" (bright on the left). Show your sum. Is the result positive, negative, or zero, and what does that mean?
2. A 6×6 feature map is max-pooled with 2×2 regions. How big is the output, and how many numbers are thrown away? What is kept?
3. You have 250 photos of road signs and want a sign reader. Write two sentences: why transfer learning beats training from scratch here, and what augmentation you would add first.

Practice quiz

1. What does a CNN filter do?

- It deletes the photo
- It slides over the image and makes a map of where its pattern appears
- It counts the pixels

- It colours the photo

Answer: It slides over the image and makes a map of where its pattern appears

2. Why is reusing the same filter everywhere useful?

- It saves memory only
- The network finds a pattern at any position with far fewer weights
- It makes photos smaller
- It removes colours

Answer: The network finds a pattern at any position with far fewer weights

3. What does max pooling do?

- It adds more pixels
- It keeps only the strongest value in each small region, shrinking the picture
- It changes colours
- It trains the network

Answer: It keeps only the strongest value in each small region, shrinking the picture

---

Watch the video lesson: https://youtu.be/Ip4mCwtFVXI

Read online: https://scopetreasure.com/notes/dl-cnns-images · More free tools and notes at https://scopetreasure.com