AI Research Labs

Section 10 of 11 · 4 lessons

Convolutional Neural Networks

Why images need a different layer, how convolution and pooling work, and how CNNs build features from edges to objects.

10.1 From Fully Connected to Convolutional

10.2 The Convolution Operation

10.3 Pooling

10.4 CNN Components and Feature Hierarchies



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

?

BUILDING AND TRAINING A NETWORK · THE FORK

Does the input have spatial structure?

Dense layers, or convolutions?

Lessons here: 10.1 · feeds into Understand the neuron, Convolve

C1

BUILDING AND TRAINING A NETWORK · CONVOLUTIONAL LANE · STEP C1

Convolve

How does a filter find a pattern anywhere in the image?

Lessons here: 10.2

C2

BUILDING AND TRAINING A NETWORK · CONVOLUTIONAL LANE · STEP C2

Pool

How do feature maps shrink while keeping what matters?

Lessons here: 10.3

C3

BUILDING AND TRAINING A NETWORK · CONVOLUTIONAL LANE · STEP C3

Stack into a hierarchy

How do layers build up from edges to objects?

Lessons here: 10.4 · feeds into Choose the output and loss, Size the network

From Fully Connected to Convolutional

Why a fully connected layer is a poor fit for images, and the two ideas that fix it: local connectivity and weight sharing. With the parameter count that shows the difference.

The problem with fully connected layers on images

In a fully connected (FC) layer every output unit connects to every input value. For an image that ignores two facts:

- **Locality:** nearby pixels are related; a pixel in the top-left corner says little about one in the bottom-right.
- **Translation:** the same pattern (an edge, an eye) can appear anywhere, and should be detected the same way wherever it is.

Two ideas

IDEA	WHAT IT MEANS	EFFECT
Local connectivity	Each output looks only at a small patch of the input (e.g. 3×3)	Far fewer connections
Weight sharing	The same small filter is applied at every position	Parameters don't depend on image size; a pattern is detected anywhere

WORKED EXAMPLE: PARAMETER COUNT

A 32×32 colour image has $32 \times 32 \times 3 = 3{,}072$ values.

- **Fully connected** to 100 units: $3{,}072 \times 100 + 100 = 307{,}300$ parameters.
- **Convolutional**, 16 filters of size 3×3×3: $16 \times 27 + 16 = 448$ parameters.

The convolutional count depends on the filter size and the number of filters, not on the number of positions the filters slide over.

Rule: a conv layer with C_{out} filters of size $k \times k$ over C_{in} input channels has $k \cdot k \cdot C_{\text{in}} \cdot C_{\text{out}} + C_{\text{out}}$ parameters.

Common doubts

FIRST PASS **Do fewer parameters make a CNN less powerful?**

Not for images. The parameters it gives up encode connections that don't help (distant pixels, position-specific copies of the same detector). Building locality and translation into the architecture makes learning far more data-efficient, which is why CNNs beat FC networks on images while being much smaller.

Cheatsheet

Why convolutions

Local connectivity (small patches) + weight sharing (same filter everywhere). 32×32×3 → 100 FC units: 307,300 params; 16 filters 3×3×3: 448.

Conv layer parameters

$k \cdot k \cdot C_{\text{in}} \cdot C_{\text{out}} + C_{\text{out}}$ (weights + one bias per filter).

Online version with the interactive quiz: airesearchlabs.si/learn/dl/convolutional-networks/from-fully-connected-to-convolutional

The Convolution Operation

Slide a small filter over the input, multiply and add at each position: that produces a feature map. A full worked example by hand, what the filter detects, channels, and the output-size formula.

The operation

A **filter** (kernel) is a small grid of learned weights. Place it over a patch of the input, multiply matching values, add them up (plus a bias): that's one output value. Slide it one step and repeat. The grid of outputs is a **feature map**.

WORKED EXAMPLE: 4×4 INPUT, 3×3 FILTER, STRIDE 1, NO PADDING

Input

1	2	0	1
0	1	3	1
2	1	0	0
1	0	1	2

$*$

Filter (vertical edges)

1	0	-1
1	0	-1
1	0	-1

$=$

Feature map

0	2
-1	-1

Top-left output: the filter covers the highlighted 3×3 patch. Column sums of the patch are $1 + 0 + 2 = 3$, $2 + 1 + 1 = 4$ and $0 + 3 + 0 = 3$; the filter weights the columns $+1, 0, -1$, so the output is $3 - 3 = 0$.

Top-right: columns 2–4 of the top three rows give sums $4, 3, 2$, so $4 - 2 = 2$. Bottom row (rows 2–4): $3 - 4 = -1$ and $2 - 3 = -1$.

This filter responds to a **vertical edge**: brighter on the left than the right gives a positive value. A CNN learns its filter values; early layers typically end up with edge- and colour-detectors like this one.

Channels

A colour image has 3 channels, so each filter is $k \times k \times 3$ and sums over all three. A layer with 16 filters produces 16 feature maps, which become 16 input channels for the next layer.

Output size

The standard formula, for input size n, filter size k, padding p and stride s:

$$\text{output size} = \left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1$$

Here $n = 4$, $k = 3$, $p = 0$, $s = 1$: $(4 - 3)/1 + 1 = 2$. **Padding** adds a border (usually zeros) so the output can keep the input's size; **stride** is the step between positions. Check your course's notation for these before relying on them in an exam.

A NAMING DETAIL

Deep learning "convolution" is, strictly, cross-correlation: the filter isn't flipped. Since the filter is learned, the difference doesn't matter in practice.

Common doubts

FIRST PASS **Is the same filter really used at every position?**

Yes. That's weight sharing: the 9 weights of a 3×3 filter (times the channels) are the only weights it has, reused at every position. Each different filter in a layer has its own weights and produces its own feature map.

Cheatsheet

Convolution

Slide filter, multiply-add per position (+ bias) → feature map. Output size $\lfloor (n + 2p - k)/s \rfloor + 1$.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/convolutional-networks/the-convolution-operation

Pooling

Pooling shrinks feature maps by summarising small regions. Max and average pooling worked out by hand, and what pooling buys: less computation and tolerance to small shifts.

Max and average pooling

Pooling slides a window (often 2×2, stride 2) over a feature map and keeps one number per window: the **maximum** or the **average**. It has no learned weights.

WORKED EXAMPLE: 2×2 POOLING, STRIDE 2

Feature map

1	3	2	0
4	2	1	5
0	1	3	2
2	6	1	1

→ Max pool

4	5
6	3

or Average pool

2.5	2
2.25	1.75

Top-left window $\{1, 3, 4, 2\}$: max 4, average 2.5. Top-right $\{2, 0, 1, 5\}$: max 5, average 2. Bottom-left $\{0, 1, 2, 6\}$: max 6, average 2.25. Bottom-right $\{3, 2, 1, 1\}$: max 3, average 1.75. The highlighted cells are the maxima.

What it buys

- **Smaller maps:** 2×2 pooling quarters the number of values, cutting computation in later layers.
- **Tolerance to small shifts:** if a feature moves by a pixel within a window, the max is unchanged. This gives a degree of translation invariance.
- Later units see a **larger region** of the original image, which helps them detect bigger structures.

Common doubts

FIRST PASS **Does max pooling throw information away?**

Yes, deliberately: it keeps whether a feature was detected strongly somewhere in the window and discards exactly where. For recognising objects that's usually what you want. For tasks needing precise positions (such as segmentation), architectures use less pooling or recover the detail later.

Cheatsheet

Pooling

Max or average over windows (e.g. 2×2, stride 2). No weights. Shrinks maps, adds tolerance to small shifts.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/convolutional-networks/pooling

CNN Components and Feature Hierarchies

The role of each part of a CNN (convolutional layers, activations, pooling and fully connected layers) and how stacking them builds features from edges to textures to whole objects.

The parts and their roles

COMPONENT	ROLE
Convolutional layers	Apply learned filters to extract local features into feature maps
Activation functions (ReLU)	Add non-linearity after each convolution, so stacked layers don't collapse into one linear map
Pooling layers	Shrink the maps, add tolerance to small shifts, cut computation
Fully connected layers	Near the output: combine the extracted features into the final prediction (with softmax for classification)

A typical stack: [convolution → ReLU → pooling] repeated a few times, then flatten, then one or two fully connected layers, then the output.

The feature hierarchy

DEPTH	TYPICAL LEARNED FEATURES
Early layers	Edges, corners, colour blobs
Middle layers	Textures, simple shapes, parts (an eye, a wheel)
Deep layers	Whole objects and faces

Each layer combines the features of the layer below, over a larger area of the image. This mirrors how images are actually built (objects from parts, parts from edges), which is why CNNs generalise well on images and why they no longer need hand-crafted features.

EXAM-READY SUMMARY

"CNNs exploit spatial locality through local connectivity and weight sharing, which cuts parameters dramatically. Stacked convolution, non-linearity and pooling layers learn a hierarchy from edges to parts to objects, and fully connected layers at the end make the prediction."

Common doubts

GOING DEEPER **How does a deep unit "see" a large part of the image if each filter is only 3×3?**

Through stacking. A unit in layer 2 sees a 3×3 patch of layer-1 maps, and each of those saw a 3×3 patch of the image, so it depends on a 5×5 region of the image. Pooling enlarges this region further. This region is called the receptive field, and it grows with depth.

Cheatsheet

CNN components

Conv: local features. ReLU: non-linearity. Pooling: shrink + shift tolerance. FC: combine features → prediction.

Feature hierarchy

Early: edges, corners. Middle: textures, parts. Deep: objects, faces. Each layer combines the one below over a larger area.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/convolutional-networks/cnn-components-and-feature-hierarchies