

Convolutional Neural Networks

How AlexNet learned to see

Mikhail Mikhasenko · Lecture 3B

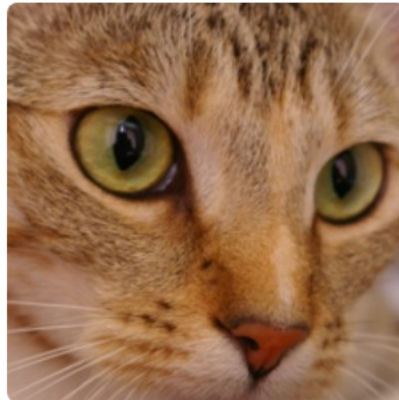
From numbers to pictures

A colour image is a **tensor** of pixel intensities: height $\times$ width $\times$ 3 channels (red, green, blue).

AlexNet reads a $224 \times 224 \times 3$ tensor, **150 528 numbers**, and returns 1000 probabilities, one per ImageNet class.

Inside there is nothing but the neurons of Lecture 3A: weighted sums, ReLUs, and a softmax at the end.

The new idea is **how the weights are arranged**.



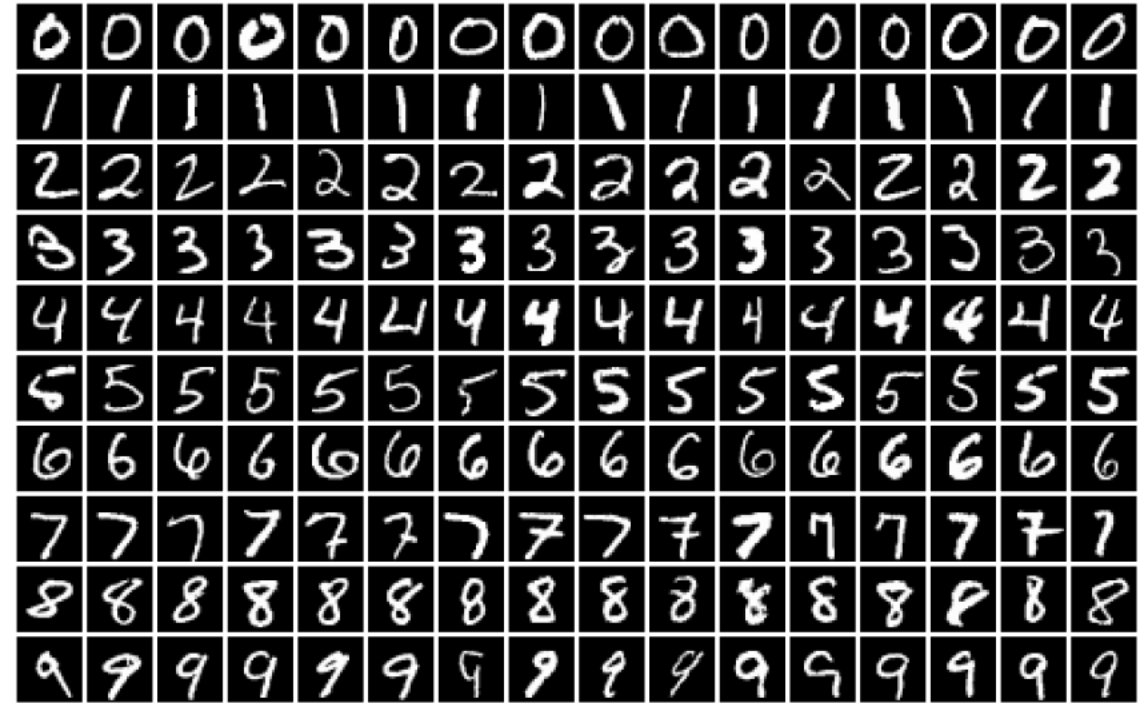
$224 \times 224 \times 3 \rightarrow \text{AlexNet} \rightarrow$

class	probability
Egyptian cat	0.80
tiger cat	0.13
tabby	0.07

MNIST: the classic benchmark

- 28 × 28 pixels, grey scale from 0 (black) to 255 (white)
- 60 000 training and 10 000 test images
- 10 classes: the digits 0–9
- Collected by LeCun, Cortes and Burges (1998) from US census and school handwriting

Small enough for a laptop, hard enough to separate methods.



MNIST: error rates

Model	Details	Test error
Linear classifier	pairwise, deskewed inputs	7.6 %
Random forest	fast unified random forests	2.8 %
Boosted stumps	on Haar features	0.87 %
Support-vector machine	degree-9 polynomial, jittered	0.56 %
Neural network	784–800–10, no distortions	1.6 %
Deep neural network	784–2500–2000–1500–1000–500–10, elastic distortions	0.35 %
Convolutional network	committee of 35 CNNs	0.23 %
Convolutional network	ensemble of 3 CNNs, augmentation	0.09 %

Dense networks need hand-made distortions to compete. Convolutional networks build in what images are like.

Why not a dense network?

Too many weights

One dense layer from a $224 \times 224 \times 3$ image to 4096 units:

$$150\,528 \times 4096 \approx 6.2 \times 10^8 \text{ weights}$$

Wrong assumptions

A dense unit treats pixel 17 and pixel 17 000 as unrelated inputs. Shift the cat by a few pixels and every input changes.

What we know about images

Locality: nearby pixels belong together; an edge is a local pattern.

Translation: a pattern means the same thing anywhere in the image.

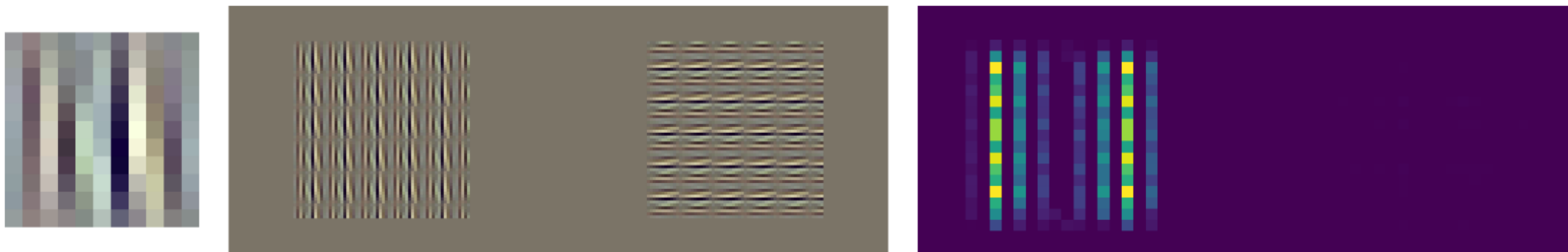
So: use **small** detectors, and apply the **same** detector at every position.

A kernel is a template

Slide a small grid of weights, the **kernel**, over the image. At each position take the **dot product** of kernel and patch:

$$s = \sum_i w_i x_i = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta$$

Large when the patch **looks like** the kernel: a similarity score.



AlexNet kernel 9 (left) slid over an image that contains its own pattern, upright and rotated by 90°. Activation map (right): peak 9.0 on the upright copy, 0.0 on the rotated copy.

Convolution

For one kernel W of size $K \times K$, bias b , stride S and padding P :

$$a_{ij} = h\left(b + \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} W_{uv} x_{iS+u-P, jS+v-P}\right)$$

Output size for an $N \times N$ input:

$$N' = \left\lfloor \frac{N + 2P - K}{S} \right\rfloor + 1$$

- **Local:** each output looks at a $K \times K$ patch.
- **Shared:** the same $K^2 + 1$ numbers are used at every position.
- **Stride** S skips positions; **padding** P adds a border of zeros.

AlexNet conv1: $\lfloor (224 + 4 - 11) / 4 \rfloor + 1 = 55$.

Deep-learning “convolution” does not flip the kernel; mathematicians call it cross-correlation.

Sliding a kernel over an image

Sliding a kernel over an image

draw on the input · shift-drag erases · click the output to jump

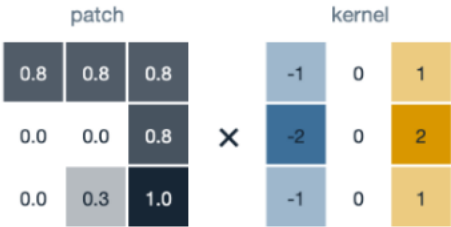
At every position the output is the dot product of the kernel with the image patch under it: a similarity score.

Image digit 7 Kernel vertical edge

-1	0	1
-2	0	2
-1	0	1

Stride 1 Padding 0 (valid) ☐ ReLU Slide Full map

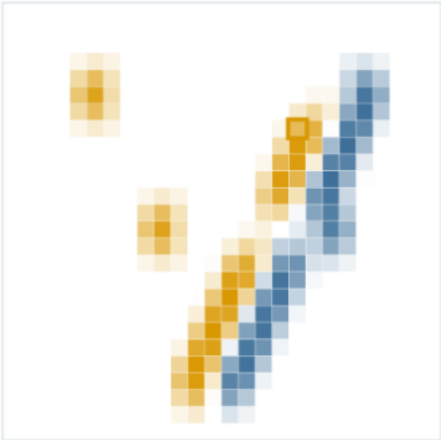
Input 28 × 28



= Σ patch × kernel = 2.58

Nine multiplications and one sum.
The same nine weights are used at every position.

Activation map 26 × 26

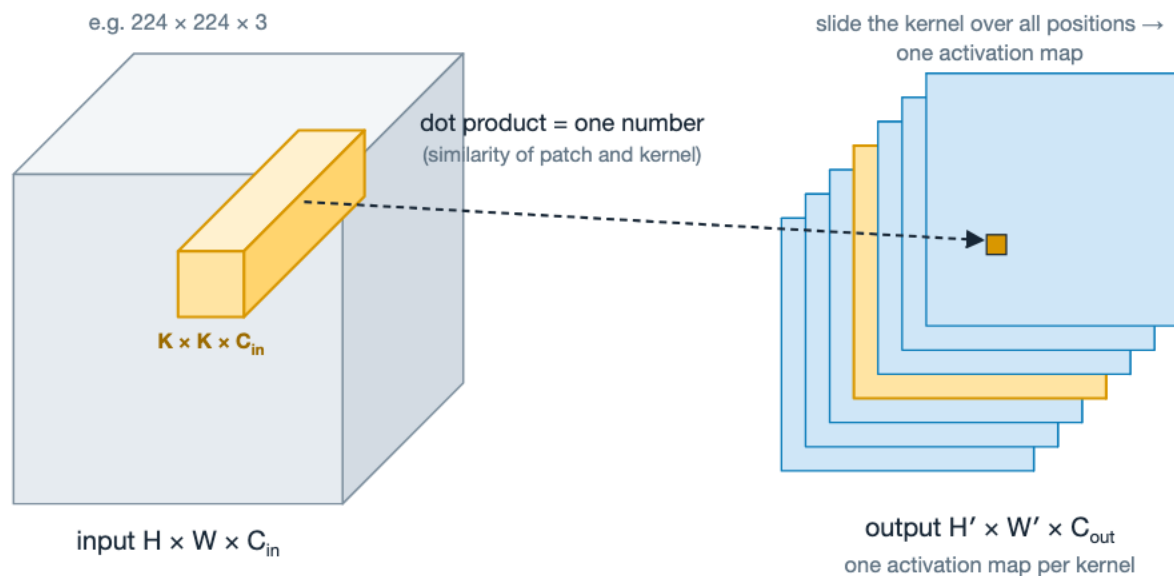


blue = negative, gold = positive

Output size	Position (row, col)	Value here	Map min / max	Weights
$\lfloor (28 + 2 \cdot 0 - 3) / 1 \rfloor + 1 = 26$	(7, 17)	2.58	-4.0 / 4.0	9 (+ 1 bias)

Vertical-edge kernel on the digit 7: the map is positive on one side of each vertical stroke, negative on the other, and zero on flat regions.

Channels: stack and repeat



A kernel always spans **all input channels**:
 $K \times K \times C_{in}$ weights.

C_{out} kernels give C_{out} activation maps.
Stacked, they form a new tensor
 $H' \times W' \times C_{out}$.

The next layer processes it exactly like an
image, only with C_{out} “colours”.

Parameters: $K^2 C_{in} C_{out} + C_{out}$.

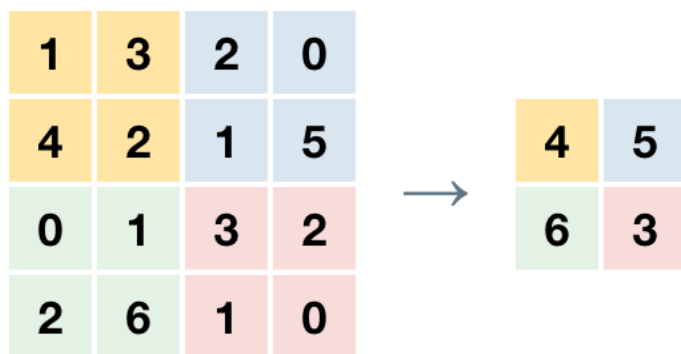
Counting parameters

	Weights	Share of AlexNet
conv1: 64 kernels of $11 \times 11 \times 3$, plus biases	23 296	
a dense layer with the same $55 \times 55 \times 64$ outputs	2.9×10^{10}	
all 5 convolutional layers	2.5 M	4 % of weights, 92 % of computation
the 3 dense layers	58.6 M	96 % of weights, 8 % of computation

Weight sharing cuts conv1 by a factor of a million. The convolutional layers are cheap to store but expensive to run: every weight is used at thousands of positions.

Pooling

Max pooling keeps the largest value in each window:



2 × 2 window, stride 2: a quarter of the values survive.

- **Smaller maps:** less computation in later layers.
- **Tolerance to small shifts:** the maximum barely moves.
- **Larger receptive fields:** later kernels see more of the image.

No weights to learn. AlexNet uses overlapping 3 × 3 windows with stride 2.

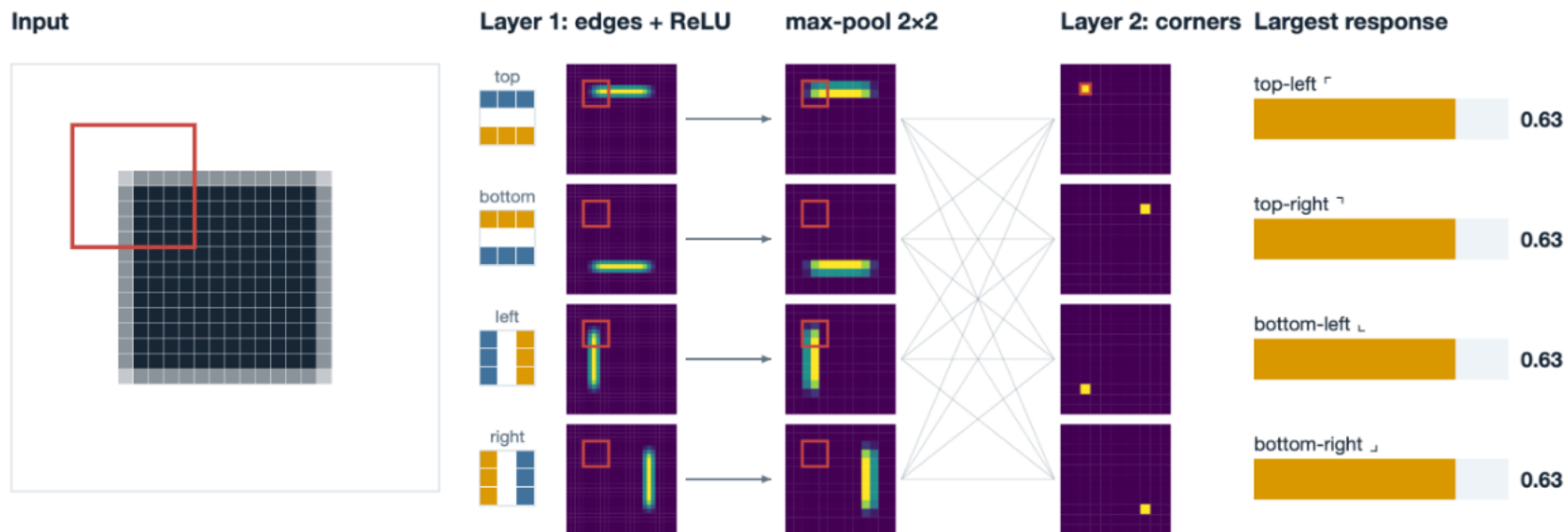
Stack and repeat: from edges to corners

Stack and repeat: from edges to corners

draw on the input · hover a corner map to see its receptive field

Layer 1 finds edges. Its four maps are stacked into a 4-channel input for layer 2, whose kernels combine two edge types into corners.

Image Layer-2 bias



Largest corner responses
0.63 0.63 0.63 0.63

Receptive field of a corner unit
8 × 8 pixels

Weights
layer 1: 4 × 9 · layer 2: 4 × (4 × 9) + 4

The square lights up exactly one unit in each corner map. Draw a new shape: which corners does it have? A plain edge is not enough to pass the bias.

A convolutional network

Blocks

$[\text{conv} \rightarrow \text{ReLU} \rightarrow \text{pool}] \times n \rightarrow \text{flatten} \rightarrow \text{dense}$
layers $\rightarrow$ softmax

Features first, then a classifier: the dense end is the MLP of Lecture 3A.

LeNet-5 (LeCun et al., 1998)

32×32 digit $\rightarrow$ 6 maps $28 \times 28 \rightarrow$ pool $\rightarrow$ 16 maps
 $10 \times 10 \rightarrow$ pool $\rightarrow$ 120 $\rightarrow$ 84 $\rightarrow$ 10. About 60 000
parameters; read bank cheques in the 1990s.

Receptive fields grow

AlexNet layer	pixels seen by one unit
conv1	11×11
conv2	51×51
conv3	99×99
conv5	163×163

Deep units see large parts of the 224×224 image.

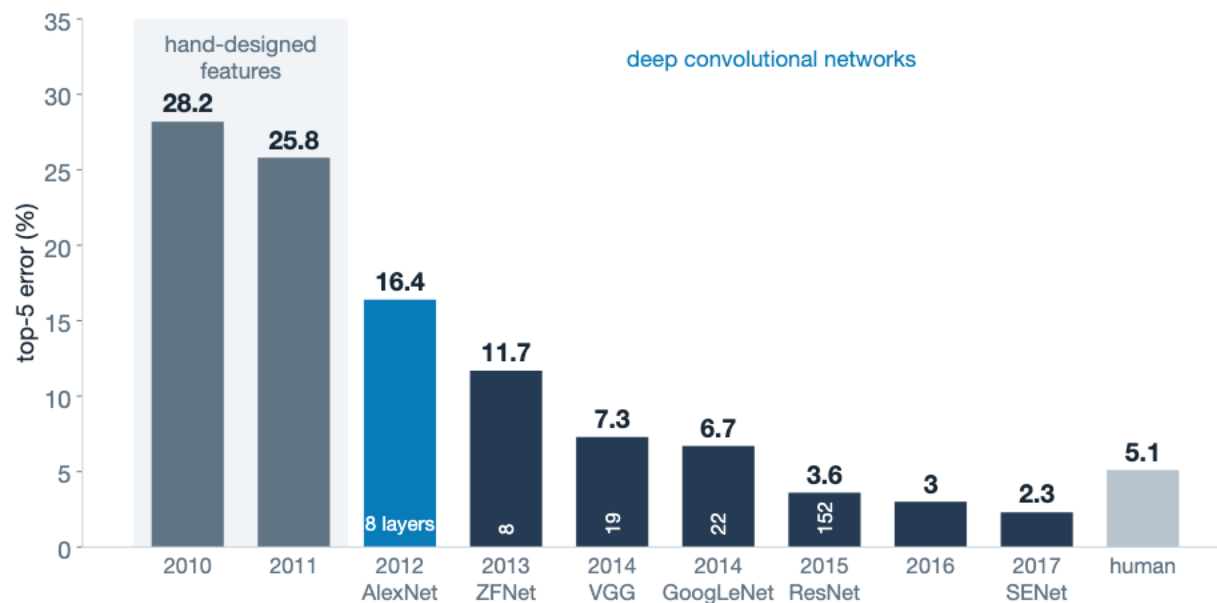
ImageNet

ILSVRC 2010–2017: 1.2 million training images, 1000 classes. Score: **top-5 error**, the fraction of images whose true class is not among the five best guesses.

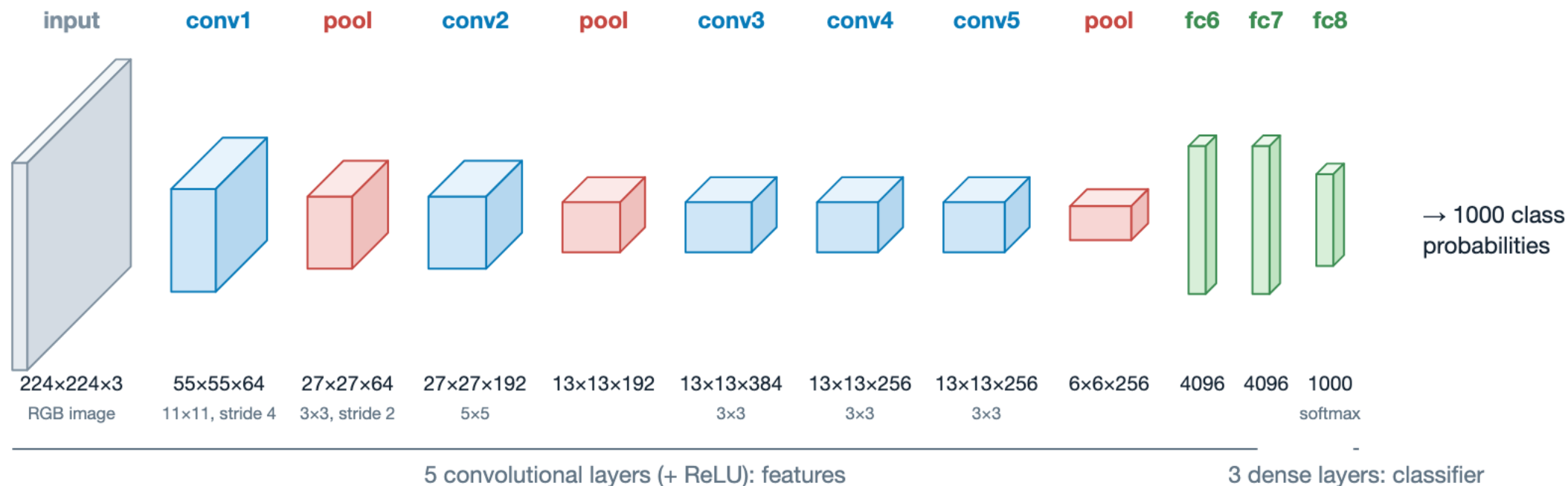
Before 2012: hand-designed features (SIFT, Fisher vectors) and linear classifiers.

2012, AlexNet (Krizhevsky, Sutskever, Hinton): 16.4 %, against 26.2 % for the runner-up.

After 2012 every winner was a deep convolutional network.



AlexNet



(CONV → RN → MP)² → CONV³ → MP → (FC → DO)² → Linear → softmax

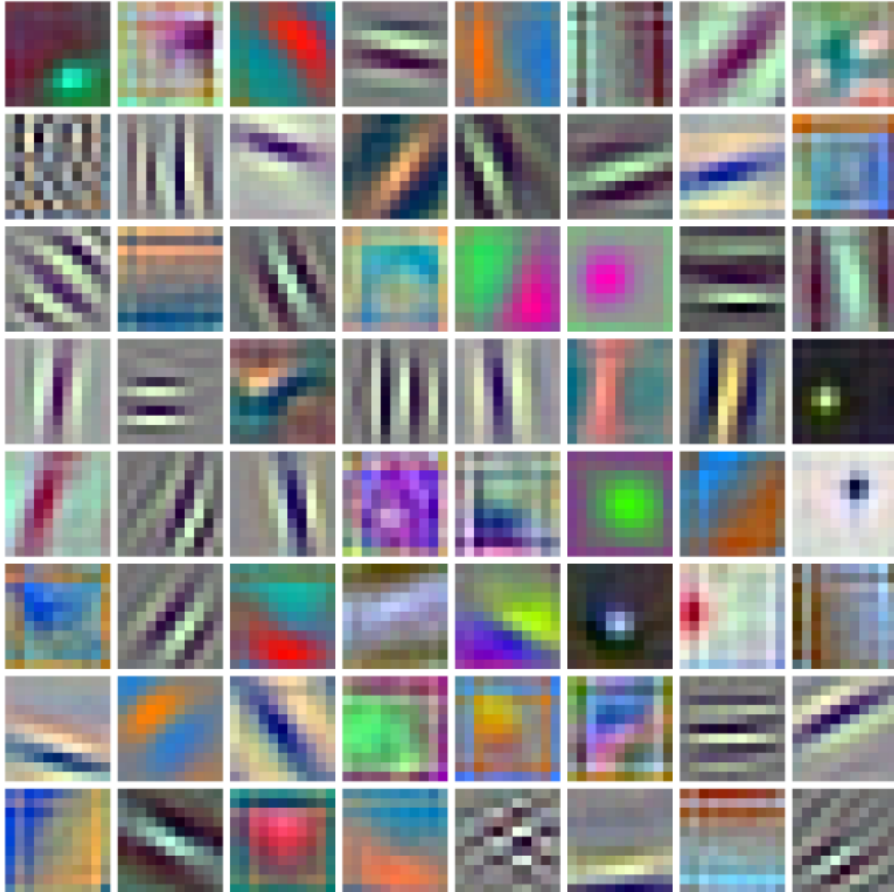
CONV: convolution + ReLU · RN: local response normalization · MP: max pooling · FC: dense + ReLU · DO: dropout

What made AlexNet work

- **ReLU** activations: several times faster training than tanh
- **Dropout** ($p = 0.5$) in the two big dense layers
- **Data augmentation**: random 224×224 crops of 256×256 images, horizontal flips, colour shifts
- SGD with momentum 0.9, weight decay 0.0005, batch 128
- **Two GTX 580 GPUs** (3 GB each), five to six days of training
- 60 million parameters, 650 000 neurons
- ImageNet: the first labelled image set large enough to feed such a model

The algorithms were from the 1980s. The **scale of data and compute** was new.

Layer 1: what did AlexNet learn?

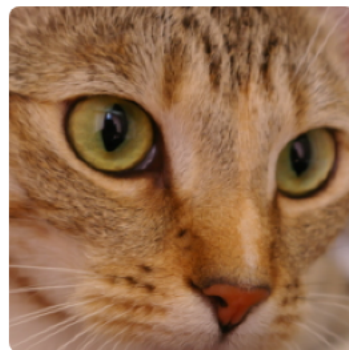
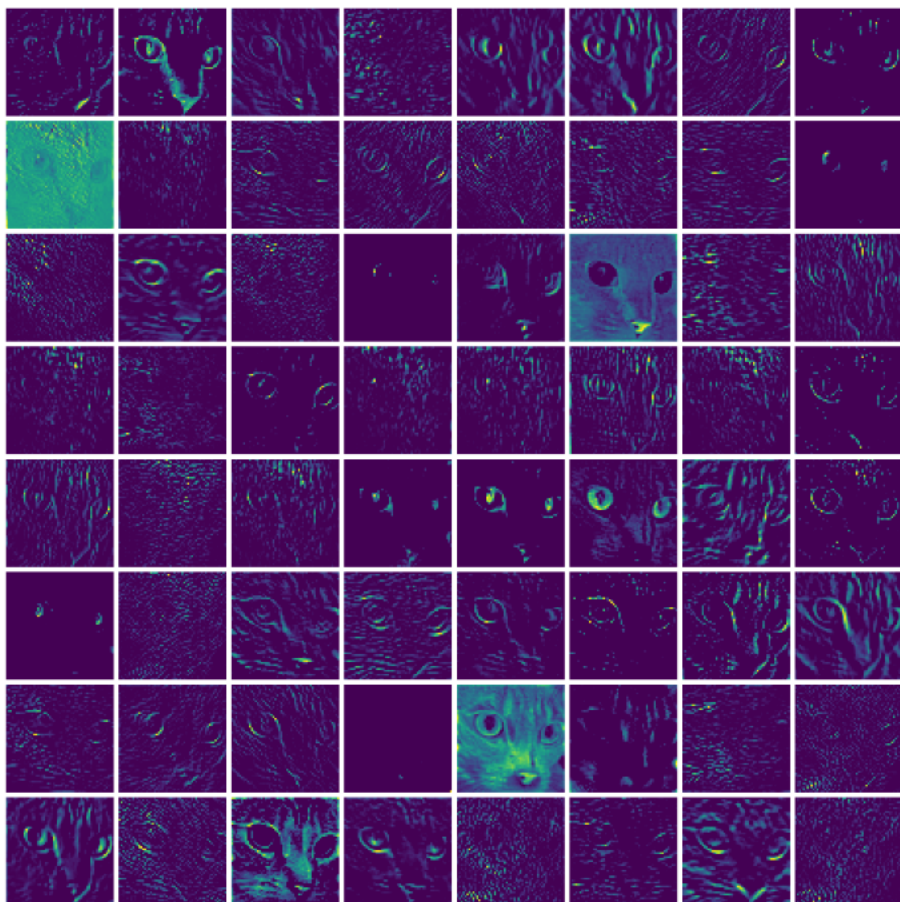


Each first-layer kernel has depth 3, like the image, so it **is** a tiny RGB picture.

- **edge detectors** at many angles and frequencies
- **colour blobs** and colour-opponent edges

Every kernel started as random numbers. These patterns were learned from labelled photos alone.

Activation maps



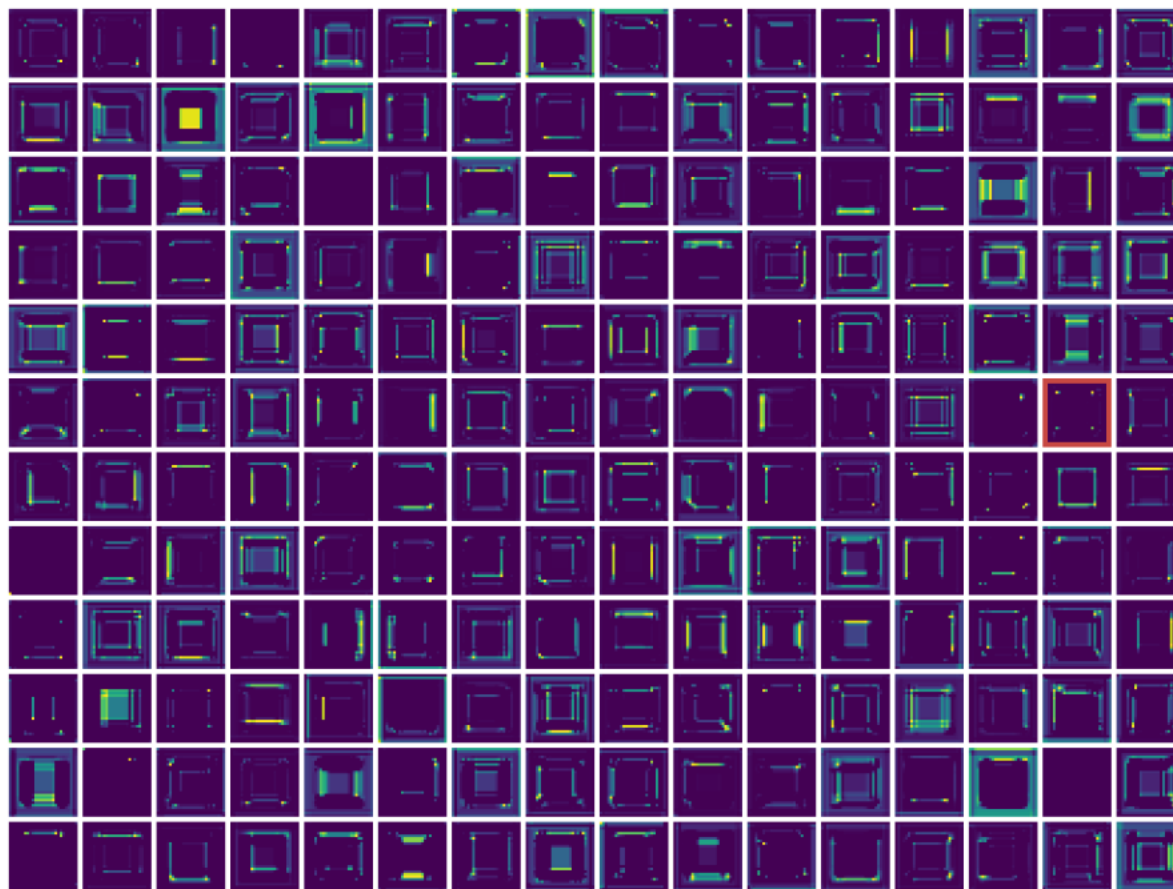
Sliding each of the 64 kernels over the photo gives 64 **activation maps** (55×55 after ReLU).

A map is bright where the image looks like its kernel: whiskers, eye outlines, fur texture, the pink nose.

Layer 2: combining edges



input: a sheet of paper



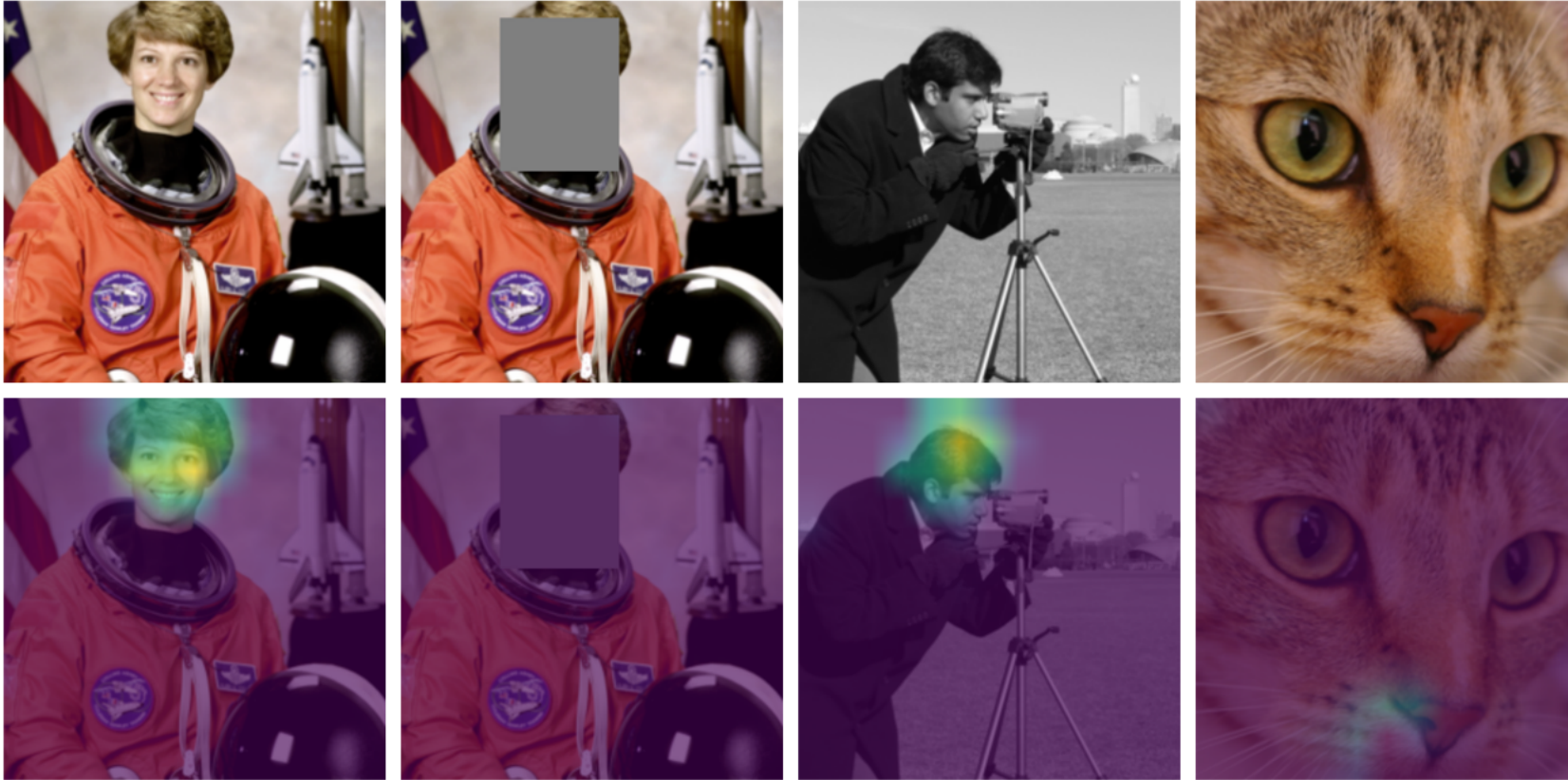
all 192 conv2 maps



channel 94: corners

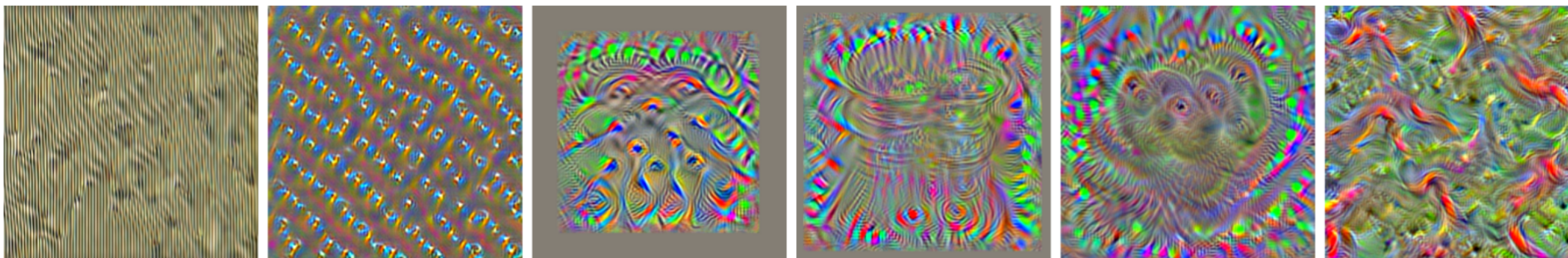
Layer-2 kernels are $5 \times 5 \times 64$: we cannot draw them, but we can watch what they respond to. Most maps trace edges of the square; channel 94 fires almost only at its **corners**.

Layer 5: a face detector nobody asked for



conv5, channel 174 · peak activation **14.5** on the astronaut's face, **0.8** with the face covered, **14.5** on the cameraman, 5.8 on the cat's nose. ImageNet has **no** "face" or "person" class.

What does a unit want to see?



conv1 · 9
stripes

conv2 · 94
corners

conv3 · 150

conv4 · 60

conv5 · 174
eyes, nose

fc8 · goldfish

Start from noise and **change the image** by gradient ascent until one unit is as active as possible; the weights stay fixed. Deeper units prefer textures, patterns, object parts.

What AlexNet says, and why

image top prediction



espresso 0.97



tripod 0.72



mosque 0.63



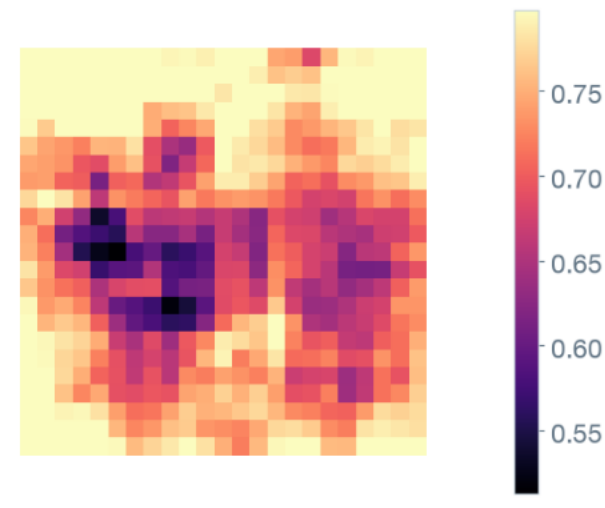
dumbbell 0.27

There is no “rocket” or “astronaut” class. A classifier always picks one of its labels.

input

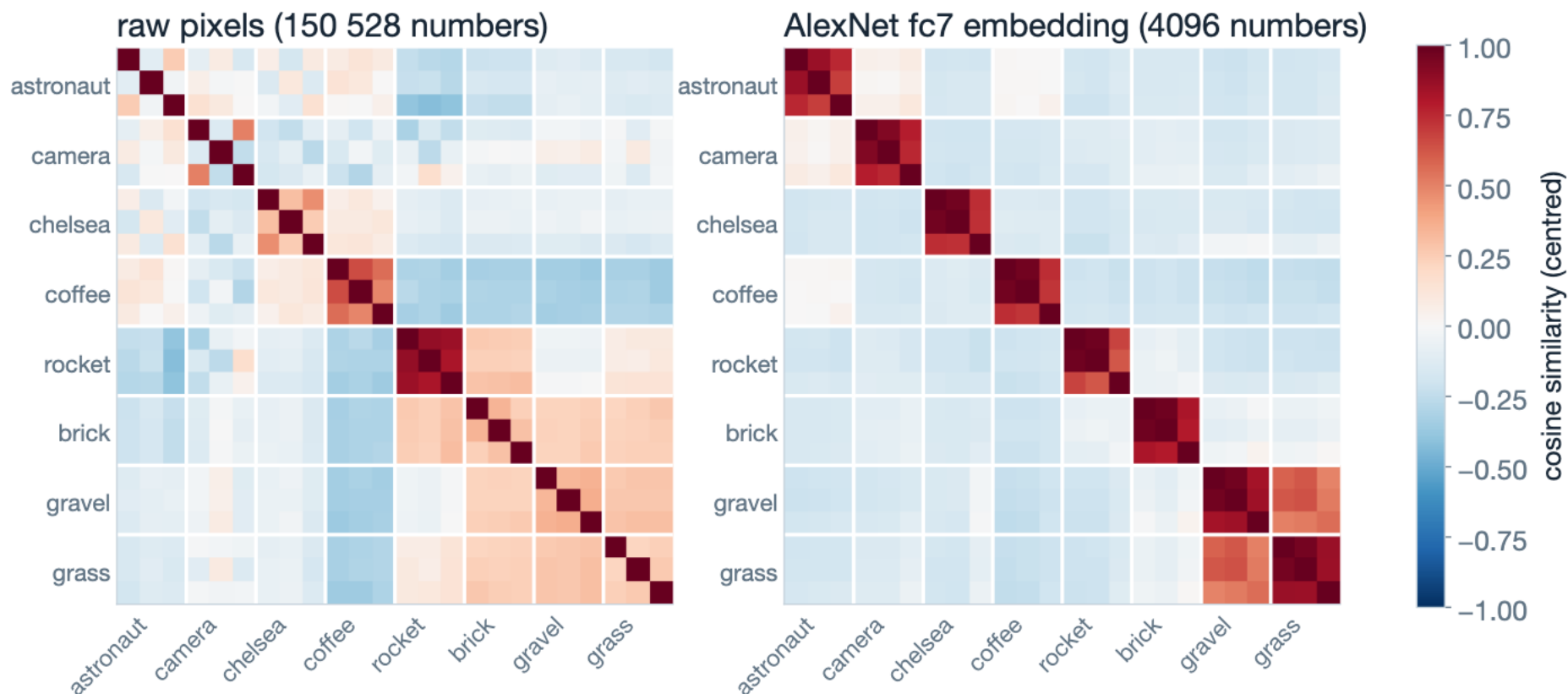


p(Egyptian cat) with a grey patch here



Occlusion: grey out a patch, record $p(\text{Egyptian cat})$. Covering the eyes hurts most ($0.80 \rightarrow 0.51$).

The embedding space



Stop before the classifier: every image becomes a point in a **4096-dimensional** space (fc7). Flipped and cropped copies of the same photo: similarity **0.34** in pixels, **0.83** in the embedding.

AlexNet under the hood: marimo notebook

`AlexNet_under_the_hood.py` loads the pretrained network and lets you

1. classify the samples or **your own photo**
2. browse the 64 layer-1 kernels and their maps
3. open the activation maps of any layer and channel
4. run the occlusion experiment
5. visualize what a unit wants to see
6. place your image in the embedding space

From the `Lecture_3B` folder:

```
1 python3.12 -m venv .venv
2 .venv/bin/python -m pip install -r requirements.txt
3 .venv/bin/marimo edit AlexNet_under_the_hood.py
```

Runs on a laptop CPU; the first start downloads the weights (~240 MB).

[Static export of the notebook](#)

After AlexNet: deeper

VGG (2014): only 3×3 kernels, 16–19 layers. Two 3×3 layers see 5×5 with fewer weights.

GoogLeNet (2014): 22 layers of parallel “inception” branches.

ResNet (2015): 152 layers, 3.6 % top-5 error. Each block learns a correction to its input:

$$\mathbf{y} = \mathbf{x} + F(\mathbf{x})$$

The skip connection gives gradients a direct path back through many layers: the vanishing-gradient problem of Lecture 3A, solved by architecture.

Since 2020, **vision transformers** cut images into patches and treat them like words, using the same blocks as language models. Their “Add & Norm” is ResNet’s skip connection.

Scale

	LeNet-5	AlexNet	GPT-3
Year	1998	2012	2020
Data	MNIST: 60 000 images, 10 classes	ImageNet: 1.2 M images, 1000 classes	~300 billion tokens of text
Compute	one CPU	two gaming GPUs	thousands of data-centre GPUs
Parameters	~60 000	~60 million	175 billion
Layers	5	8	96
Key parts	convolutions, tanh	convolutions, ReLU, dropout	transformer blocks with dense layers

The same building blocks, scaled up by factors of a thousand at every step.

End

Thank you for the attention