IMPERIAL

Convolutional Neural Networks (CNNs)

Payam Barnaghi
Department of Brain Sciences &
School of Convergence Science (Human and Artificial Intelligence)
Imperial College London
January 2026

ML4NS Machine Learning for Neuroscience

1

Feature Vectors

IMPERIAL

- Image data is represented as a two-dimensional grid of pixels, monochromatic or colour.
- Each pixel corresponds to one or multiple numerical values.
- Until now, most of the models that we have studied have overlooked this structure and treated the data as vectors of numbers.

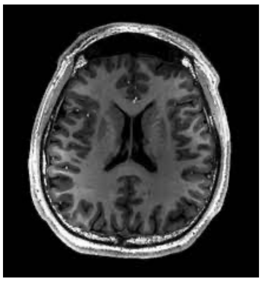
2

2

Images a vectors

IMPERIAL

- Flattening images overlooks the spatial relations between pixels.
- Although somewhat unsatisfying, this approach offers a simple way to feed the resulting one-dimensional vectors into a fully connected MLP or other probabilistic models.
- However, the spatial relation between the pixels is not explicitly included in the model.



3

3

Feeding images to the models as blocks

IMPERIAL

- Because the MLP networks are invariant to the order of the features, we could get similar results regardless of whether we preserve an order corresponding to the spatial structure of the pixels or if we permute the columns of our design matrix before fitting the MLP's parameters.
- Preferably, we would leverage our prior knowledge that nearby pixels are typically related to each other to build efficient models for learning from image data.

4

4

IMPERIAL

Convolutional Neural Networks

- Convolutional Neural Networks, or CNNs, are specialised neural networks that process data with a known grid-like topology.
- Examples include time series data, which can be thought of as a 1-D grid taking samples at regular intervals, and image data, which can be thought of as a 2-D grid of pixels.

5

5

IMPERIAL

CNNs

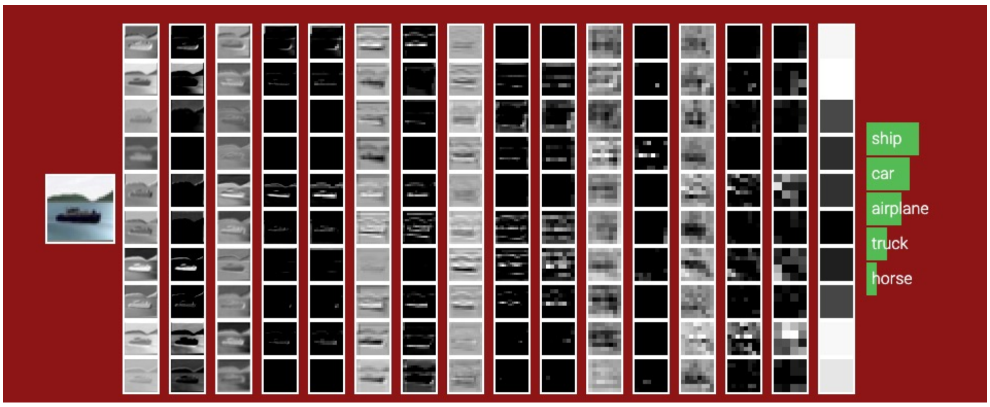


image source: http://cs231n.stanford.edu/

We will revisit this again in the following slides.

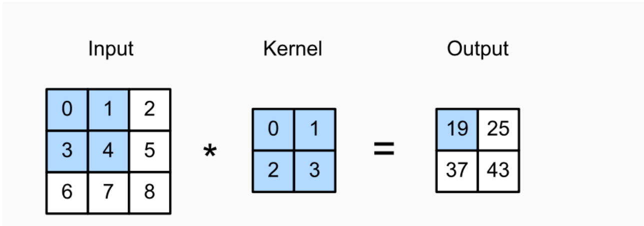
6

6

Convolution layer

IMPERIAL

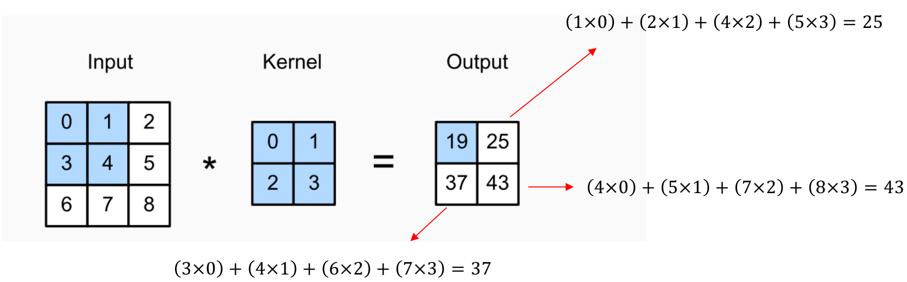
- Convolutional layers in CNN can be more accurately described as cross-correlations.
- They take an input (usually a grid or a subset of the main image) and apply a kernel.
- The shape of the *kernel window* (or *convolution window*) is given by the height and width of the kernel.



Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

CNN kernels

IMPERIAL



Two-dimensional cross-correlation operation. The shaded portions are the first output element, as well as the input and kernel tensor elements used for the output computation:

(0×0) + (1×1)+ (3×2)+ (4×3)=19.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

IMPERIAL

Sliding the kernel

$$
\begin{array}{ccccc}
\text{Input} & & \text{Kernel} & & \text{Output} \\
\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} & * & \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} & = & \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}
\end{array}
$$

$$
\begin{aligned}
0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19, \\
1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25, \\
3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37, \\
4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.
\end{aligned}
$$

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

9

9

IMPERIAL

Example – edge detection: data

```
tensor([[1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.]])
```

```
X = torch.ones((6, 8))
X[:, 2:6] = 0
X
```

Code: GitHub - PyTorch CNN_edge_detection_sample.ipynb

10

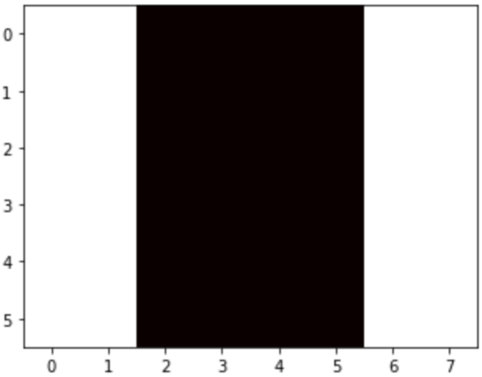
10

Example – edge detection: data

IMPERIAL

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
plt.imshow(X, cmap=plt.cm.hot)
plt.show()
```

✓ 0.1s




```
[[1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.]]
```

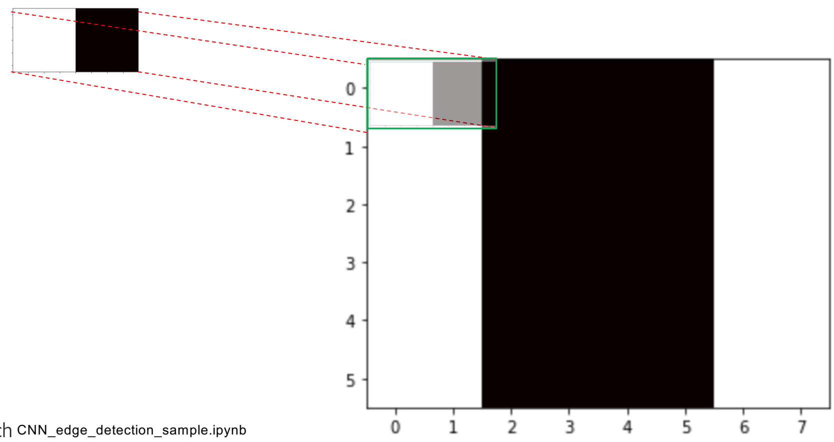
11

11

Example – edge detection: kernel

IMPERIAL

```
K = torch.tensor([[1.0, -1.0]])
```



Code: GitHub - PyTorch CNN_edge_detection_sample.ipynb

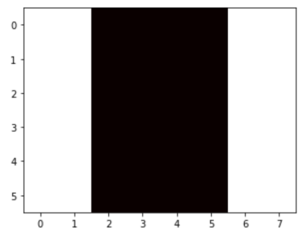
12

12

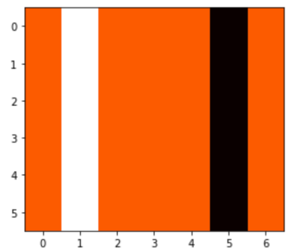
Example – edge detection: result

IMPERIAL

```
[[1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.],
 [1., 1., 0., 0., 0., 0., 1., 1.]]
```




```
[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
[ 0.,  1.,  0.,  0.,  0., -1.,  0.]]
```



Code: GitHub - PyTorch CNN_edge_detection_sample.ipynb

13

13

Learning a kernel

IMPERIAL

- Designing an edge detector by finite differences `[1,-1]` is neat if we know this is precisely what we are looking for.
- However, as we consider larger kernels and successive layers of convolution, it might be impossible to specify precisely what each filter should do manually.

14

14

Learning a kernel through training

IMPERIAL

- We can learn the kernel that generated `Y` from `X` by looking only at the input-output pairs. We first construct a convolutional layer and initialise its kernel as a random tensor.
- Next, in each iteration, we will use the squared error to compare `Y` with the convolutional layer output.
- We can then calculate the gradient to update the kernel.

15

15

Example: how to learn a kernel

IMPERIAL

```
# Construct a two-dimensional convolutional layer with 1 output channel and a
# kernel of shape (1, 2). For the sake of simplicity, we ignore the bias here
# adapted from https://d2l.ai/chapter_convolutional-neural-networks/conv-layer.html

conv2d = nn.LazyConv2d(1, kernel_size=(1, 2), bias=False)

# The two-dimensional convolutional layer uses four-dimensional input and
# output in the format of (example, channel, height, width), where the batch
# size (number of examples in the batch) and the number of channels are both 1
X = X.reshape((1, 1, 6, 8))
Y = Y.reshape((1, 1, 6, 7))
lr = 3e-2  # Learning rate

for i in range(10):
    Y_hat = conv2d(X)
    l = (Y_hat - Y) ** 2
    conv2d.zero_grad()
    l.sum().backward()
    # Update the kernel
    conv2d.weight.data[:] -= lr * conv2d.weight.grad
    if (i + 1) % 2 == 0:
        print(f'epoch {i + 1}, loss {l.sum():.3f}')
```
✓ 0.6s

```
epoch 2, loss 1.373
epoch 4, loss 0.240
epoch 6, loss 0.044
epoch 8, loss 0.009
epoch 10, loss 0.002
```

Code: GitHub - PyTorch CNN_edge_detection_sample.ipynb

16

16

Example: how to learn a kernel: result

IMPERIAL

```
[24]  X
      ✓ 0.3s

...   tensor([[[[1., 1., 0., 0., 0., 0., 1., 1.],
                [1., 1., 0., 0., 0., 0., 1., 1.],
                [1., 1., 0., 0., 0., 0., 1., 1.],
                [1., 1., 0., 0., 0., 0., 1., 1.],
                [1., 1., 0., 0., 0., 0., 1., 1.],
                [1., 1., 0., 0., 0., 0., 1., 1.]]]])
```

```
[25]  Y
      ✓ 0.3s

...   tensor([[[[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
                [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
                [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
                [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
                [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
                [ 0.,  1.,  0.,  0.,  0., -1.,  0.]]]])
```

```
[23]  conv2d.weight.data
      ✓ 0.6s

...   tensor([[[[ 0.9975, -0.9905]]]])
```

17

17

CNNs so far

IMPERIAL

- The core computation required for a convolutional layer is a cross-correlation operation.
- We saw that a simple nested for-loop is all that is required to compute its value.
- If we have multiple input and output channels, we perform a matrix-matrix operation between channels.

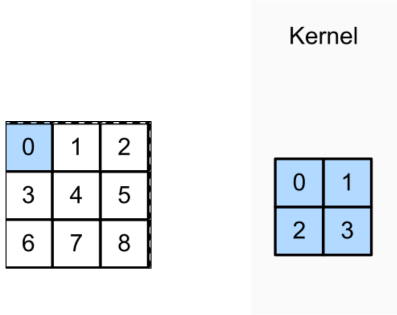
18

18

Padding and Stride

IMPERIAL

- A tricky issue when applying convolutional layers is that we tend to lose pixels on the perimeter of our image.



Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

19

19

Padding

IMPERIAL

- Since we typically use small kernels for any given convolution, we might only lose a few pixels, but this can add up as we apply many successive convolutional layers.
- One straightforward solution to this problem is to add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.
- Typically, we set the values of the extra pixels to zero.

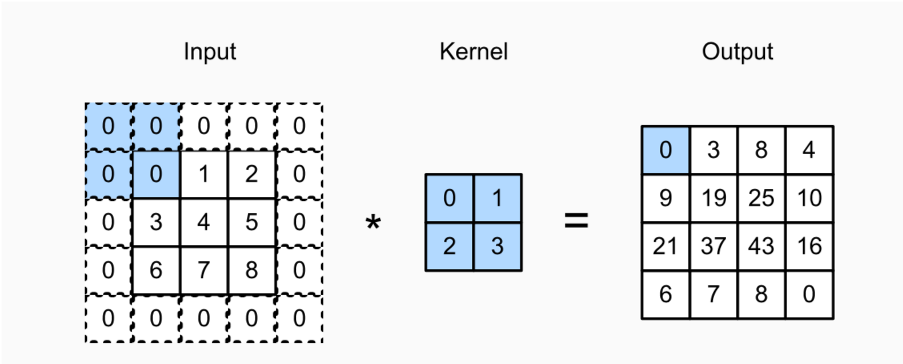
Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

20

20

Padding: example

IMPERIAL



Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Choice of padding size

IMPERIAL

- CNNs commonly use convolutional kernels with odd heights and widths, such as `1`, `3`, `5`, or `7`.
- Choosing odd kernel sizes has the benefit that we can preserve the dimensionality while padding with the same number of rows on top and bottom and the same number of columns on left and right.
- For any two-dimensional tensor `X`, when the kernel's size is odd and the number of padding rows and columns on all sides are the same, an output with the same height and width as the input is produced.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Stride

IMPERIAL

- When computing the cross-correlation, we start with the convolution window at the upper-left corner of the input tensor and then slide it over all locations both down and to the right.
- We have been sliding one element at a time in the previous examples.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Stride

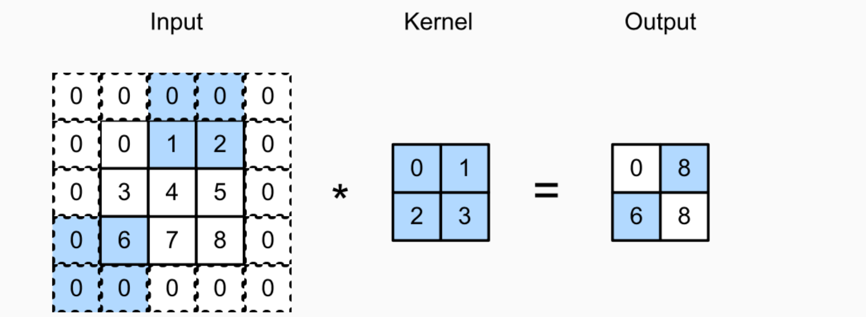
IMPERIAL

- Sometimes, either for computational efficiency or because we wish to downsample, we move our window more than one element at a time, skipping the intermediate locations.
- This is particularly useful if the convolution kernel is large since it captures a large area of the underlying image.
- We refer to the number of rows and columns traversed per slide as *stride*.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Stride: example

IMPERIAL



Cross-correlation with strides of 3 and 2 for height and width, respectively.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Choice of padding

IMPERIAL

- Padding can increase the height and width of the output.
- This is often used to ensure the output has the same height and width as the input, preventing undesirable shrinkage.
- Moreover, it ensures that all pixels are used equally frequently.
- Typically, we pick symmetric padding on both sides of the input height and width.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Multiple Input and Multiple Output Channels

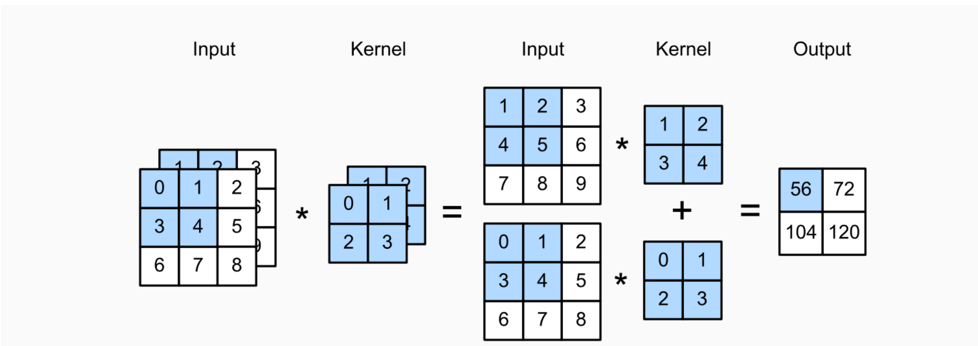
IMPERIAL

- Multiple channels can comprise each image (e.g., colour images use the standard RGB channels to represent red, green, and blue values for each pixel), and we can use convolutional layers across multiple channels.
- When the input data contains multiple channels, we need to construct a convolution kernel with the same number of input channels as the input data so that it can perform cross-correlation with the input data.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Multiple Input Channels: example

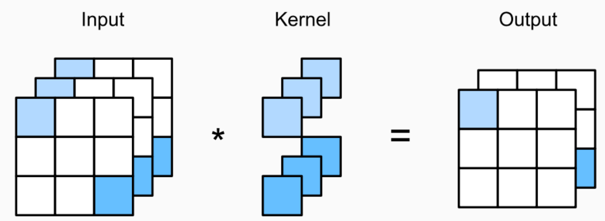
IMPERIAL



Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

Multiple Output Channel: example (1x1 convolution)

IMPERIAL



The cross-correlation computation uses the 1×1 convolution kernel with 3 input channels and 2 output channels. The input and output have the same height and width.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

29

29

CNN Layers

IMPERIAL

- Channels allow us to combine the best of both worlds: MLPs, which support significant nonlinearities, and convolutions, which enable *localised* analysis of features.
- Different layers of convolution and changes in resolution (via pooling layers) allow the CNN to reason across multiple features, such as edge and shape detectors.
- CNN models also offer a practical trade-off between the drastic parameter reduction arising from translation invariance and locality and the need for expressive and diverse models in computer vision.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

30

30

Pooling

IMPERIAL

- In many cases our ultimate task asks some global question about the image, e.g., *does it contain a lesion?*
- Consequently, the units of our final layer should be sensitive to the entire input.
- By gradually aggregating information, yielding coarser and coarser maps, we accomplish this goal of ultimately learning a global representation, while keeping all the advantages of convolutional layers at the intermediate layers of processing.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

31

31

CNN – deep layers

IMPERIAL

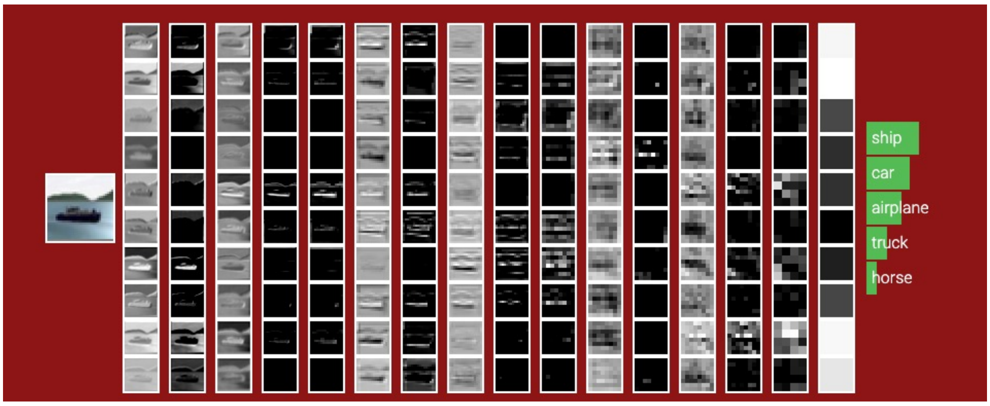


image source: http://cs231n.stanford.edu/

32

32

Deeper layers in CNNs

IMPERIAL

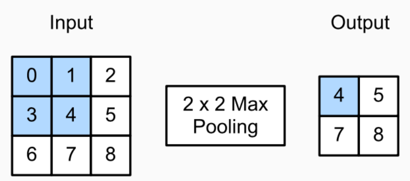
- The deeper we go in the network, the larger the receptive field (relative to the input) each hidden node is sensitive to.
- Reducing spatial resolution accelerates this process since the convolution kernels cover a larger effective area.
- In early layers, kernels learn simple patterns such as edges and corners.
- In deeper layers of a multilayer CNN, kernels learn more abstract patterns, such as shapes, objects, or specific parts of objects.
- Overall, convolution layers capture increasingly abstract features layer by layer.
- Pooling layers help the model to be more robust to minor input variations.

33

33

Maximum Pooling and Average Pooling

IMPERIAL



Max-pooling with a pooling window shape of 2×2. The shaded portions are the first output element as well as the input tensor elements used for the output computation: $\max(0, 1, 3, 4) = 4$.

Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

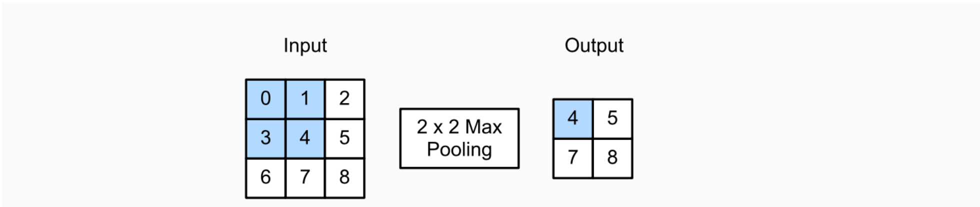
34

34

Pooling

IMPERIAL

- Like convolutional layers, *pooling* operators consist of a fixed-size window that is slid over all regions of the input according to its stride, computing a single output for each location traversed by the window (sometimes called the *pooling window*).



Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

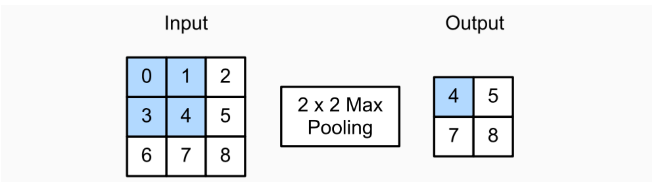
35

35

Max-pooling and average-pooling

IMPERIAL

- However, unlike the cross-correlation computation of inputs and kernels in the convolutional layer, the pooling layer contains no parameters (i.e., no *kernel*).
- Instead, pooling operators are deterministic, typically calculating either the maximum or the average value of the elements in the pooling window.
- These operations are called *maximum pooling* (*max-pooling* for short) and *average pooling*, respectively.



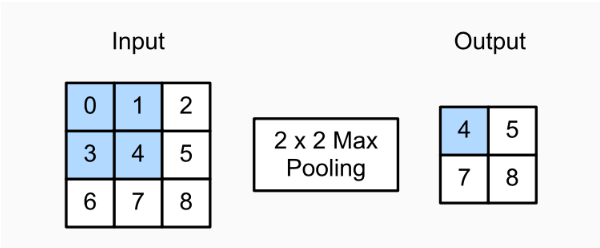
Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

36

36

Max-pooling: revisiting the example

IMPERIAL



$$\max(0,1,3,4) = 4,$$
$$\max(1,2,4,5) = 5,$$
$$\max(3,4,6,7) = 7,$$
$$\max(4,5,7,8) = 8.$$

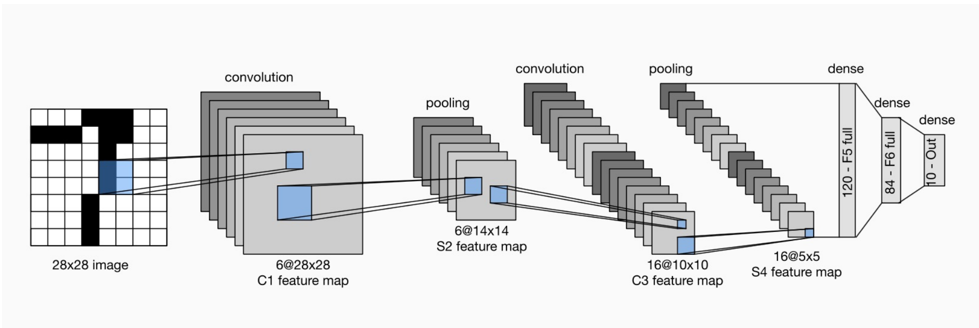
Source: Dive into Deep Learning, Aston Zhang *et al.*, https://d2l.ai

37

37

Convolutional Neural Networks: example LeNet

IMPERIAL

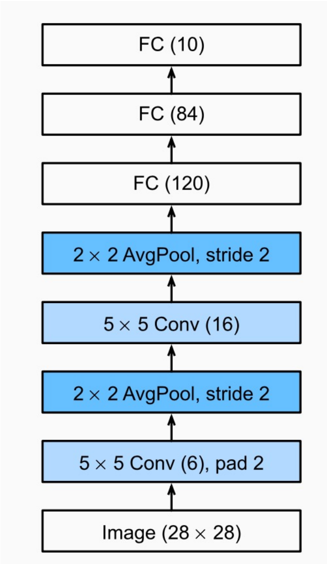


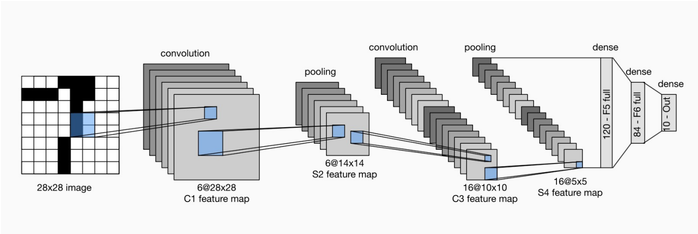
38

38

Compressed notion for LeNet-5

IMPERIAL





39

39

Example of a practical model: VGG-19

IMPERIAL

- In practice, you use models such as VGG;
  - VGG: Very Deep Convolutional Networks for Large-Scale Image Recognition by Karen Simonyan, Andrew Zisserman, ICLR 2015, https://arxiv.org/pdf/1409.1556

```
Input: 224×224 RGB Image
↓
[Conv3-64] → [Conv3-64] → [MaxPool]
↓
[Conv3-128] → [Conv3-128] → [MaxPool]
↓
[Conv3-256] → [Conv3-256] → [Conv3-256] → [Conv3-256] → [MaxPool]
↓
[Conv3-512] → [Conv3-512] → [Conv3-512] → [Conv3-512] → [MaxPool]
↓
[Conv3-512] → [Conv3-512] → [Conv3-512] → [Conv3-512] → [MaxPool]
↓
[FC-4096] → [FC-4096] → [FC-1000] → [Softmax]
```

Note: [Conv3-64] denotes a convolutional layer with a 3×3 kernel and 64 filters; [MaxPool] denotes a max-pooling layer; [FC-4096] denotes a fully connected layer with 4096 units.

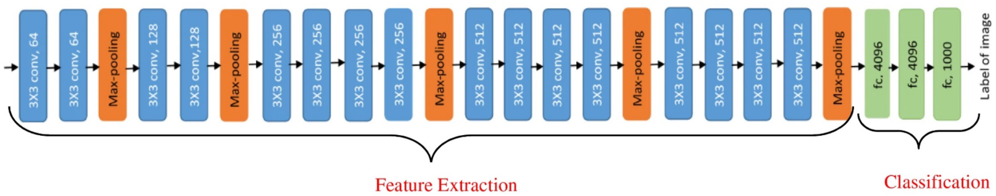


Image source: Bansal, M., Kumar, M., Sachdeva, M. *et al.* Transfer learning for image classification using VGG19: Caltech-101 image data set. *J Ambient Intell Human Comput* **14**, 3609–3620 (2023). https://doi.org/10.1007/s12652-021-03488-z

40

40

CNN Autoencoder

IMPERIAL

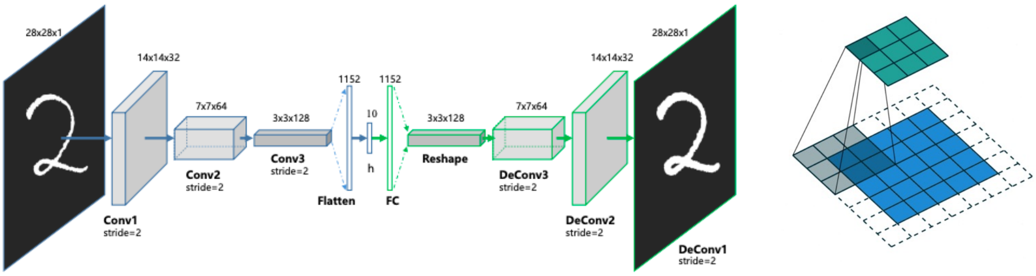


image source: https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763

41

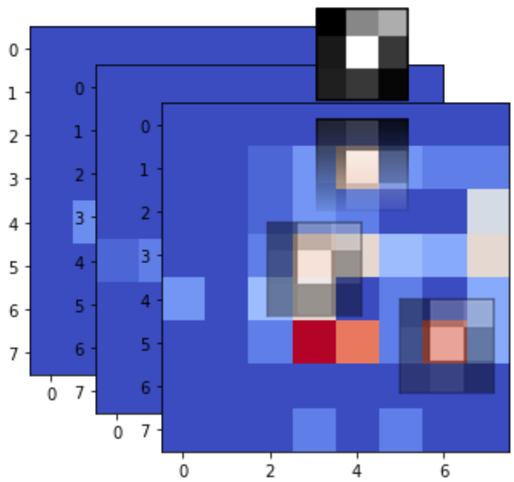
41

More examples

IMPERIAL

A sample filter which mainly focuses on the middle and top right areas out of data.

A sample filter which focuses almost on an L shape area (not equally) from the data.

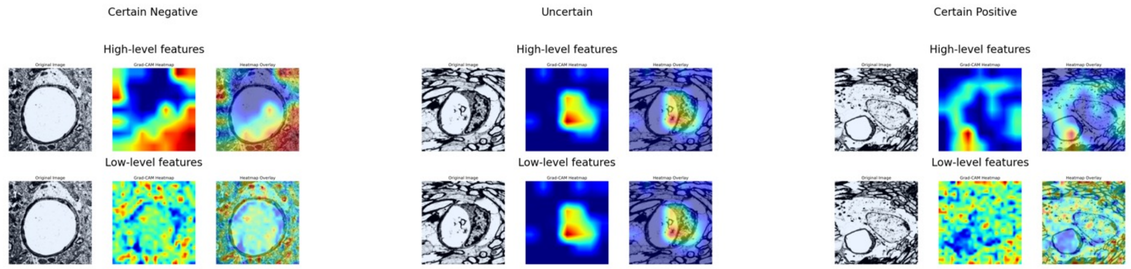


42

42

CNN Example: VGG model for cell microscopy data analysis

IMPERIAL



Nan Fletcher-Lloyd *et al.*, https://www.nature.com/articles/s42003-025-08453-6

43

43

Revisiting the initial example

IMPERIAL

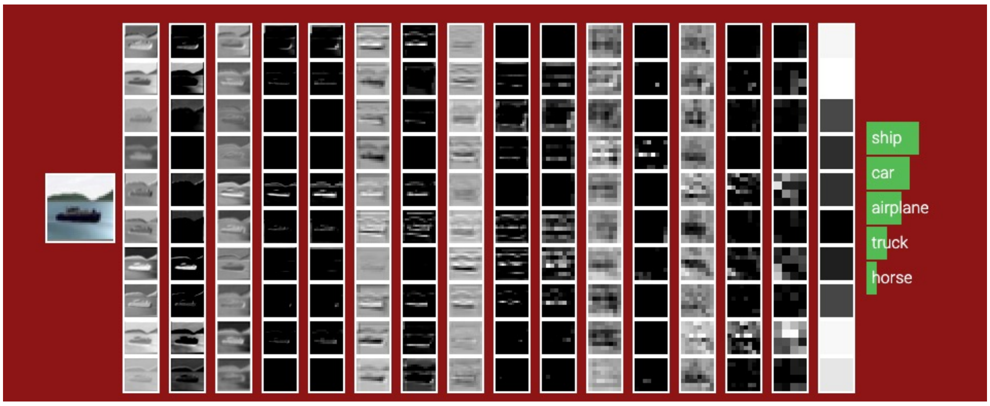


image source: http://cs231n.stanford.edu/

44

44

Open discussion

IMPERIAL

- A research team is developing a convolutional neural network (CNN) to automatically detect brain tumours from MRI scans. The dataset includes thousands of images, but tumour shapes and sizes vary widely, and some scans have noise or artefacts.
  - What makes CNNs suitable for image-based tasks compared to MLPs?
- **Architecture Choices:**
  - Should we use a simple CNN or a deeper architecture like VGG or a simpler CNN?
  - How do pooling layers affect localisation of small tumours?
- **Data Challenges:**
  - How do we handle class imbalance (few tumour cases vs many normal scans)?
  - Should we use data augmentation (rotations, flips) for MRI images?
- **Clinical Implications:**
  - What are the risks of false negatives in tumour detection?
- **Explainability:**
  - How can techniques like Grad-CAM help clinicians trust CNN predictions?
  - Should interpretability be prioritised over accuracy in medical imaging?

45

45

IMPERIAL

Review questions

46

46

Q1

In a CNN network, what technique has been used if a method generates results shown in section (b) from the data grid shown in section (a)?

12	10	3	2
8	6	1	0
4	7	5	7
4	6	6	8

(a)

12	3
7	8

(b)

Q2

In a CNN network, if we have the kernel shown in (a) and want to apply to the data shown (b) with a stride of 2, what padding size would you recommend?

1	-1
-1	1

(a)

12	10	3	2
8	6	1	0
4	7	5	7
4	6	6	8

(b)

Q3

IMPERIAL

- Which operation reduces the spatial dimensions of an image in CNNs?
  A) Convolution
  B) Pooling
  C) Padding
  D) Fully connected layer

49

49

Q4

IMPERIAL

- What does "padding" in CNNs achieve?
  A) Reduces the computational cost of the network
  B) Prevents overfitting during training
  C) Maintains the spatial dimensions of the input
  D) Enhances gradient descent optimisation

50

50

Q5

IMPERIAL

- What is the receptive field in CNNs?
  A) The total number of filters used in a layer
  B) The region of the input image that affects a single output value
  C) The activation map produced by a convolution
  D) The number of the kernel applied during convolution

51

51

If you have any questions

IMPERIAL

- Please feel free to arrange a meeting or email (p.barnaghi@imperial.ac.uk).
- To arrange a meeting, please email my colleague, Ms Rhiannon Kirby.
- My office: 928, Sir Michael Uren Research Hub, White City Campus.

52

52