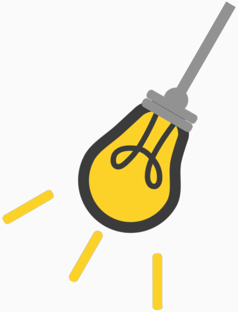
浙江大学爱丁堡大学联合学院
ZJU-UoE Institute

Lecture 12 - Convolutional Neural Networks (CNN)

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

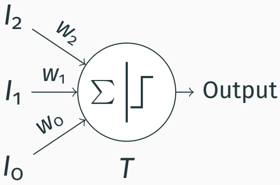
- Explain the motivation behind the use of convolutional neural networks for image analysis.
- Describe the main component of a CNN
- Describe the basic structure of a CNN
- Calculate the layer size and number of trainable parameters in a CNN



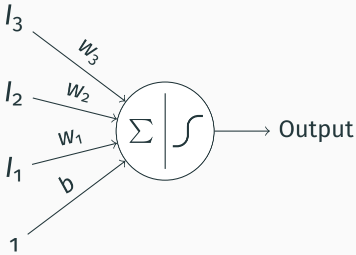
Introduction

In Lecture 11 we introduced neural networks. With increasing depth, neural networks can solve more complex problems. This comes at the cost of increased computational complexity (more parameters to learn).

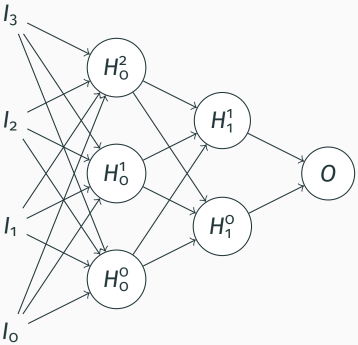
McCulloch-Pitts neuron



Single layer perceptron



Multi-layer perceptron



Neural networks for image analysis

Can we use a MLP to analyze images?

- Input image: shape(w, h, c)
- Linearize image: shape(w x h x c)
- Use this vector as input to MLP
- Train and predict

Any problem with this?

Can we use a MLP to analyze images?

- Input image: shape(w, h, c)
- Linearize image: shape(w x h x c)
- Use this vector as input to MLP
- Train and predict

Any problem with this?

- **It is impractical** for anything other than extremely small images.
  A small 256 x 256 RGB image gives 256 x 256 x 3 = 196608 inputs. Add a few hidden layers and the number of parameters to estimate becomes unmanageable.

Can we use a MLP to analyze images?

- Input image: shape(w, h, c)
- Linearize image: shape(w x h x c)
- Use this vector as input to MLP
- Train and predict

Any problem with this?

- **It is impractical** for anything other than extremely small images.
  A small 256 x 256 RGB image gives 256 x 256 x 3 = 196608 inputs. Add a few hidden layers and the number of parameters to estimate becomes unmanageable.
- MLP are not **translation invariant**.
  If our network learns to detect a cell in the top-left part of the image, it won't be able to detect it in the bottom-right part.

Can we use a MLP to analyze images?

- Input image: shape(w, h, c)
- Linearize image: shape(w x h x c)
- Use this vector as input to MLP
- Train and predict

Any problem with this?

- **It is impractical** for anything other than extremely small images.
  A small 256 x 256 RGB image gives 256 x 256 x 3 = 196608 inputs. Add a few hidden layers and the number of parameters to estimate becomes unmanageable.
- MLP are not **translation invariant**.
  If our network learns to detect a cell in the top-left part of the image, it won't be able to detect it in the bottom-right part.
- **We lose spatial information** when we flatten the image.
  Closeby pixels are more similar to each other than they are to the rest of the image. Problem for all other ML methods we have seen so far.

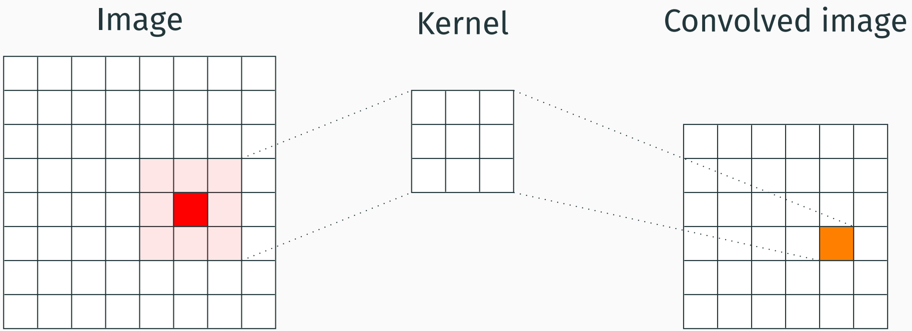
The solution?

Use convolutional filters!

(and have a neural network decide which to use!)

Convolutional filters

Convolutional filters



$$O = \sum_i \sum_j I_{i,j} K_{i,j}$$

- A 3x3 filter only has 9 parameters to learn, independently of image size.
- Convolutional filters are translation invariant.
- Convolution retains spatial information.

Some CNN convolution terminology

In CNN we define convolutional filters using:

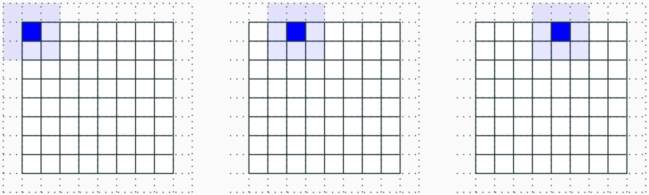
- **Stride**: the number of pixels to skip between each filter application. Strides greater than 1 **reduce the size of the output**.
- **Padding**: the number of pixels to add to the input image to make it divisible by the filter size. Normally, zero-padding is used in CNN.

Some CNN convolution terminology

In CNN we define convolutional filters using:

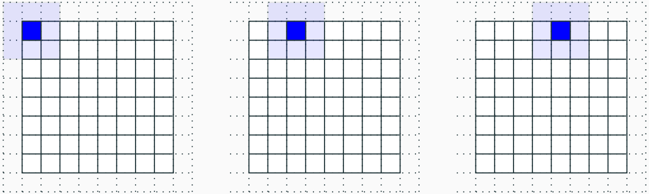
- **Stride**: the number of pixels to skip between each filter application. Strides greater than 1 **reduce the size of the output**.
- **Padding**: the number of pixels to add to the input image to make it divisible by the filter size. Normally, zero-padding is used in CNN.
- Some CNN terminology related to padding:
  - **Same padding**: we pad with the same amount of zeros on each side. If we use a stride of 1 we will have the same image shape after convolution.
  - **Valid padding**: only **valid** data is used, meaning no padding. The output image is smaller than the input image, since we cannot process edge pixels.

3 x 3 kernel, stride: 2, same padding

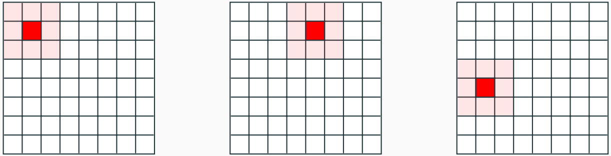


Stride and padding - examples

3 x 3 kernel, stride: 2, same padding



3 x 3 kernel, stride: 3, valid padding



Convolved image size

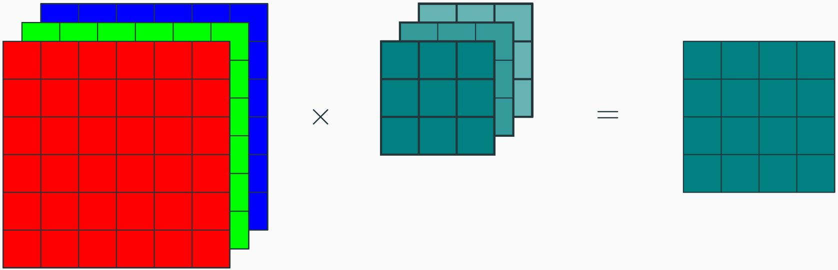
Given an image of size $n \times n$, a filter of size $k \times k$, stride s and padding p the output image size is:

$$\left\lfloor \frac{n + 2p - k}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - k}{s} + 1 \right\rfloor$$

Convolution of a volume

When applying convolution to a volume, we need to do it with a 3D filter.

For example, we can convolve an RGB image with a $3 \times 3 \times 3$ filter.

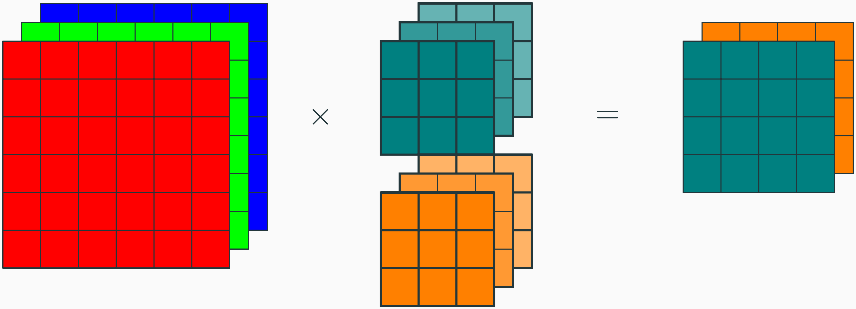


6×6 RGB image $3 \times 3 \times 3$ filter 4×4 image

Convolution of a volume

When applying convolution to a volume, we need to do it with a 3D filter.

For example, we can convolve an RGB image with a $3 \times 3 \times 3$ filter.



6×6 RGB image   Two $3 \times 3 \times 3$ filters $4 \times 4 \times 2$ image

The general idea behind convolutional neural networks

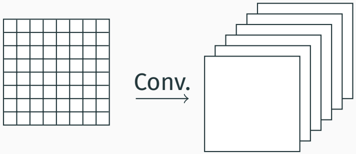
The general idea behind convolutional neural networks

1. We start by training the network on a set of images.
2. We update the filter weights using back-propagation (e.g. gradient descent).
3. We can use the trained network on a new set of images!

Building a CNN

Convolutional layer

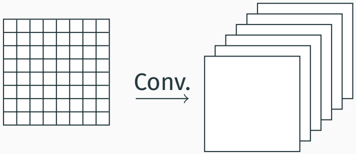
- Arguably the most important part of a CNN



Convolutional layer, 6 convolutions

Convolutional layer

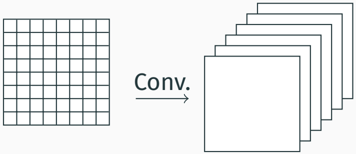
- Arguably the most important part of a CNN
- Performs a series of convolutions on the input image.



Convolutional layer, 6 convolutions

Convolutional layer

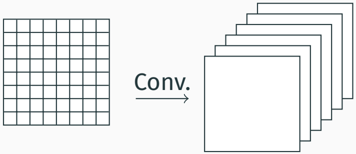
- Arguably the most important part of a CNN
- Performs a series of convolutions on the input image.
- Hyperparameters: number of convolutions, filter size, stride, padding.
  Note: the size, stride and padding are the same for all convolutions in the same layer.



Convolutional layer, 6 convolutions

Convolutional layer

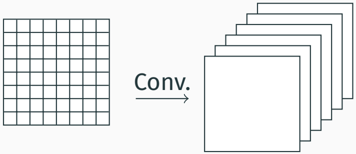
- Arguably the most important part of a CNN
- Performs a series of convolutions on the input image.
- Hyperparameters: number of convolutions, filter size, stride, padding.
  Note: the size, stride and padding are the same for all convolutions in the same layer.
- Parameters to learn: filter weights and biases.



Convolutional layer, 6 convolutions

Convolutional layer

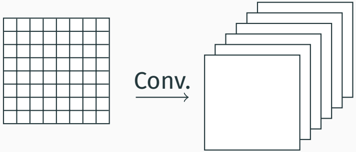
- Arguably the most important part of a CNN
- Performs a series of convolutions on the input image.
- Hyperparameters: number of convolutions, filter size, stride, padding.
  Note: the size, stride and padding are the same for all convolutions in the same layer.
- Parameters to learn: filter weights and biases.
- Number of parameters: num filters $\times$ filter size $+ 1$ (bias).



Convolutional layer, 6 convolutions

Convolutional layer

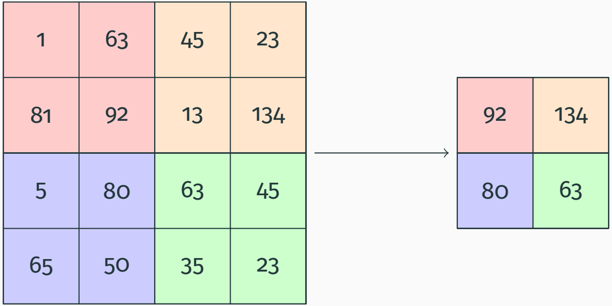
- Arguably the most important part of a CNN
- Performs a series of convolutions on the input image.
- Hyperparameters: number of convolutions, filter size, stride, padding.
  Note: the size, stride and padding are the same for all convolutions in the same layer.
- Parameters to learn: filter weights and biases.
- Number of parameters: num filters $\times$ filter size $+ 1$ (bias).
- After convolution a non-linearity is introduced through the **activation function**. **ReLU** is the most commonly used.



Convolutional layer, 6 convolutions

Max-pooling layer

- Performs a maximum filter on the input.



Max-pooling layer

- Performs a maximum filter on the input.
- Hyperparameters: filter size, stride (padding usually 0).

1	63	45	23
81	92	13	134
5	80	63	45
65	50	35	23

$\rightarrow$

92	134
80	63

Pooling layers - max pooling

Max-pooling layer

- Performs a maximum filter on the input.
- Hyperparameters: filter size, stride (padding usually 0).
- Parameters to learn: none.

1	63	45	23
81	92	13	134
5	80	63	45
65	50	35	23

→

92	134
80	63

Pooling layers - max pooling

Max-pooling layer

- Performs a maximum filter on the input.
- Hyperparameters: filter size, stride (padding usually 0).
- Parameters to learn: none.
- Mostly used after a convolutional layer (thus some people call "convolutional + max pooling" a layer, others will consider them two layers, there is no consensus).

1	63	45	23
81	92	13	134
5	80	63	45
65	50	35	23

$\longrightarrow$

92	134
80	63

Pooling layers - average pooling

Average-pooling layer

- Performs average filtering on the input.

1	63	45	23
81	92	13	134
5	80	63	45
65	50	35	23

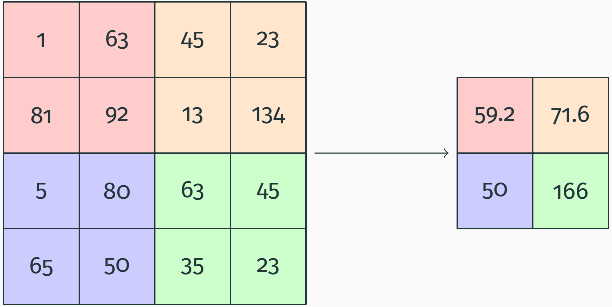
$\longrightarrow$

59.2	71.6
50	166

Pooling layers - average pooling

Average-pooling layer

- Performs average filtering on the input.
- Hyperparameters: filter size, stride (padding usually 0).



Average-pooling layer

- Performs average filtering on the input.
- Hyperparameters: filter size, stride (padding usually 0).
- Parameters to learn: none.

1	63	45	23
81	92	13	134
5	80	63	45
65	50	35	23

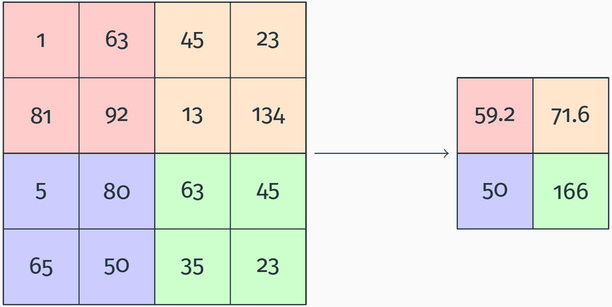
$\longrightarrow$

59.2	71.6
50	166

Pooling layers - average pooling

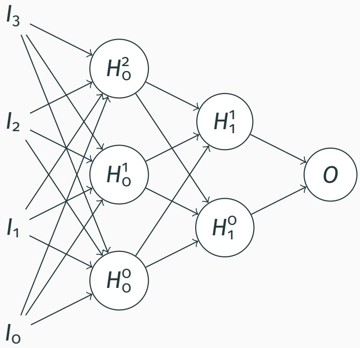
Average-pooling layer

- Performs average filtering on the input.
- Hyperparameters: filter size, stride (padding usually 0).
- Parameters to learn: none.
- Rarely used nowadays



Eventually, after all the convolutions we will need to flatten our output and feed it to a **fully-connected layer** (that is, a **multi-layer perceptron**!).

This will take care, for example, of the final classification task.



Putting it all together

We are now ready to put everything together.

Example, we want to create a CNN that can classify an image as one of two classes (e.g. illness vs no illness).

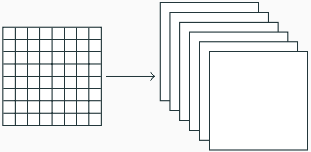


Image
$\mathbf{32 \times 32}$

6 filters,
size 5,
s=1, p=0
$\mathbf{28 \times 28 \times 6}$

Putting it all together

We are now ready to put everything together.

Example, we want to create a CNN that can classify an image as one of two classes (e.g. illness vs no illness).

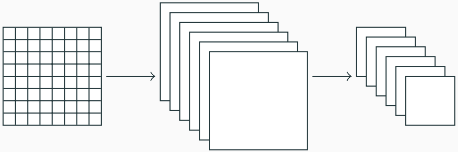


Image	6 filters, size 5, s=1, p=0	2x2 max pooling, s=2
$\mathbf{32 \times 32}$	$\mathbf{28 \times 28 \times 6}$	$\mathbf{14 \times 14 \times 6}$

Putting it all together

We are now ready to put everything together.

Example, we want to create a CNN that can classify an image as one of two classes (e.g. illness vs no illness).

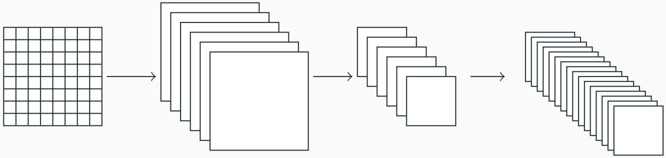


Image
32 $\times$ **32**

6 filters,
size 5,
s=1, p=0
28 $\times$ **28** $\times$ **6**

2x2 max
pooling, s=2
14 $\times$ **14** $\times$ **6**

16 filters,
size 5, s=1
10 $\times$ **10** $\times$ **16**

Putting it all together

We are now ready to put everything together.

Example, we want to create a CNN that can classify an image as one of two classes (e.g. illness vs no illness).

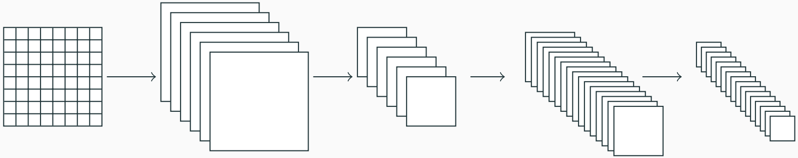


Image	6 filters, size 5, s=1, p=0	2x2 max pooling, s=2	16 filters, size 5, s=1	2×2 max pooling, s=2
$\mathbf{32 \times 32}$	$\mathbf{28 \times 28 \times 6}$	$\mathbf{14 \times 14 \times 6}$	$\mathbf{10 \times 10 \times 16}$	$\mathbf{5 \times 5 \times 16}$

Putting it all together

We are now ready to put everything together.

Example, we want to create a CNN that can classify an image as one of two classes (e.g. illness vs no illness).

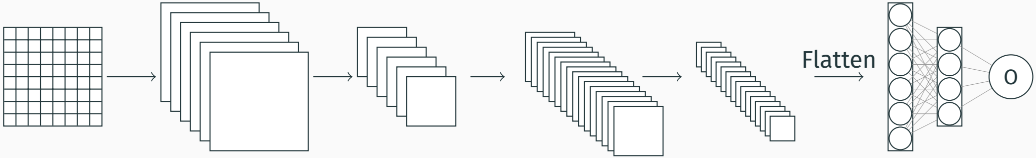


Image
32 × 32

6 filters,
size 5,
s=1, p=0
28 × 28 × 6

2x2 max
pooling, s=2
14 × 14 × 6

16 filters,
size 5, s=1
10 × 10 × 16

2 × 2 max
pooling, s=2
5 × 5 × 16

Fully
connected
(100 + 50
nodes)

Summary of our CNN

	Activation shape	Activation size	Number of params
Input	(32,32)	1024	-
Convolution $6 \times$ (f=5, s=1, p=0)	(28,28,6)	4704	150 $(5 \times 5 + 1) * 6$
Max pooling (f=2, s=0)	(14,14,6)	1176	0
Convolution $16 \times$ (f=5, s=1, p=0)	(10,10,16)	1600	2416 $(5 \times 5 \times 6 + 1) * 16$
Max pooling (f=2, s=0)	(5,5,16)	400	0
Fully connected	(100,1)	100	40100 $(400 \times 100 + 100)$
Fully connected	(50,1)	50	5050 $(100 \times 50 + 50)$
Output	(1,1)	1	50
		TOTAL	**47766 parameters**

Coming up...

In the next lectures we will explore some "classic" CNN structures, which you can use as a starting point for your own CNNs.

We will talk about the practical use of the CNNs for image analysis, e.g. for segmentation.

We will also build our own!