Image classification with Convolutional Neural Networks

Lecture 5

Course of:
Signal and imaging acquisition and modelling in environment

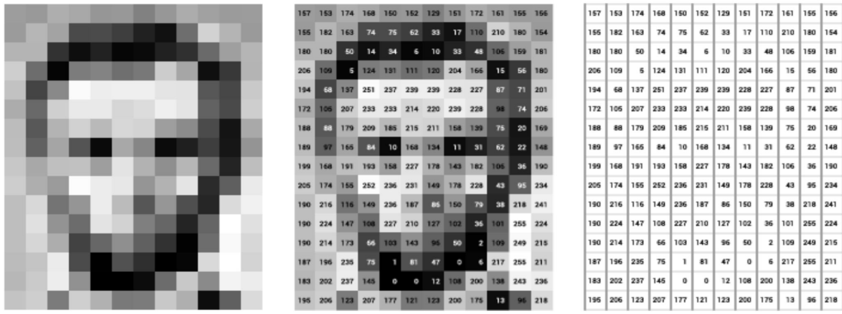
20/03/2024

Federico De Guio - Matteo Fossati

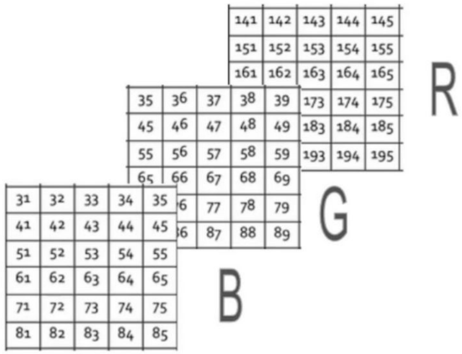
Digital images (reminder)

- From Wikipedia: "A digital image is an image **composed of picture elements, also known as pixels, each with finite, discrete quantities of numeric representation for its intensity.**"
    - Images can be **greyscale**, where each pixel value is the grey intensity, or **colored**

- Convolutional Neural Networks are a powerful family of neural networks that are specifically designed for the **Digital Images Processing Task**



Colored images



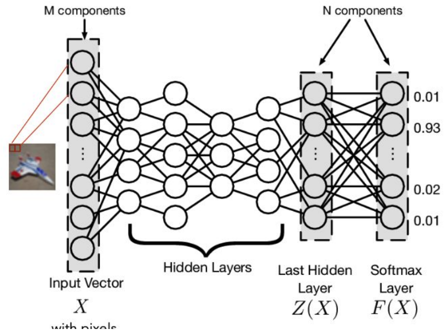
- Colored images are usually coded with the **RGB color model**: each pixel is associated to three numbers, corresponding to the **Red, Green and Blue intensity**

- An RGB image is therefore represented by a matrix (weight)x(height)x3
- A greyscale image is (weight)x(height)x1

Our goal:

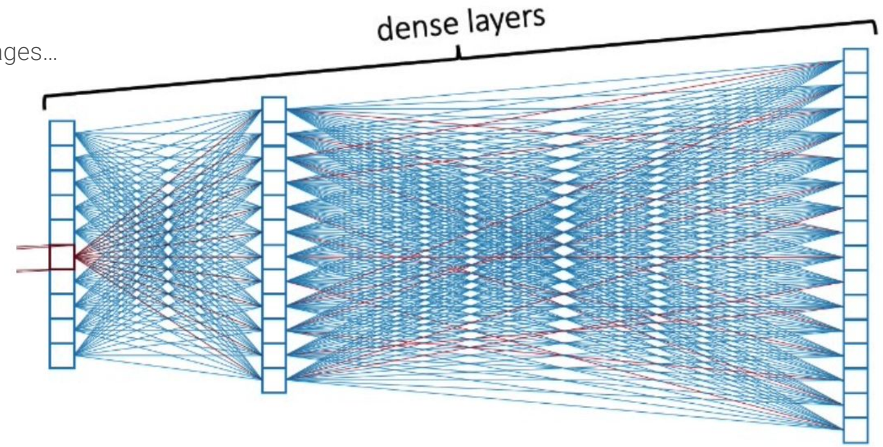
Classify images using ML

Images classification with DNN



- In principle **a FF DNN could be used to classify images**
  - Flatten the image into a 1D vector of pixels

- In practice, if we take a **64x64 grayscale** image:
  - 1D vector of 4096 values → **4096 nodes in the input layer**
  - if the (fully connected) inner layer has **500 nodes**, we will have 4096x500 = **2048000 weights** between the input and the hidden layer
  - Account for a factor 3 for colored images…



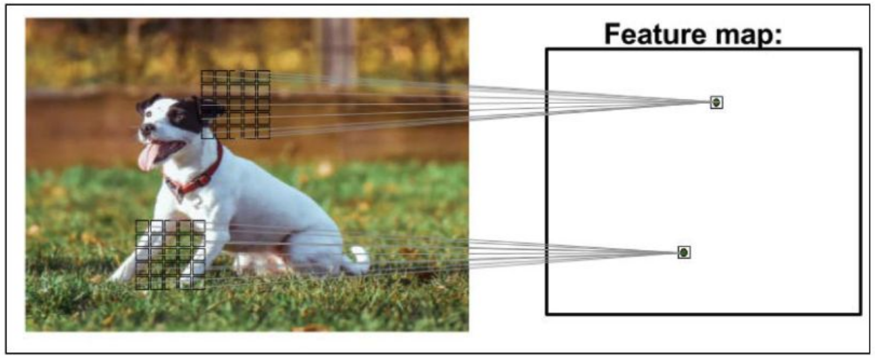
DOES NOT SCALE!!!

Convolutional Neural Networks (CNN)

- To flat an image into a 1D array is not the best way to model images
  - any **spatial relationship in the data is ignored**

- A CNN **maintains the spatial structure of the data**, and is better suited for finding spatial relationships in the image data

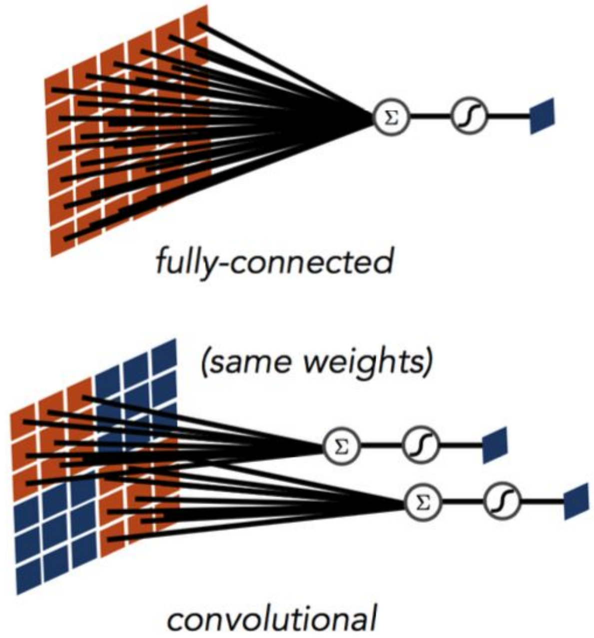
- **The idea behind:** use filters that automatically learns the most discriminants features in an image, such as edges, filled patterns, specific geometric forms and so on…



Convolutional Neural Networks (CNN)

- **Basic idea:** assume that features are **translation invariant**
  - → share the parameters across (x, y)

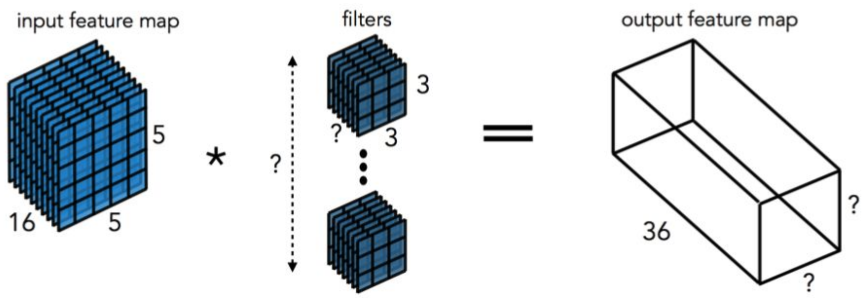
- **Advantages**:
  - Save a lot of parameters
  - fast, highly **parallelizable** on GPUs



The convolutional layer

- It is the core **building block of a Convolutional Network**

- The CONV layer's parameters consist of a **set of learnable filters**.
- Each filter outputs an activation map → **Conv layer outputs a volume**



With: stride=1 and padding=0

1. How many filters are there?
36

2. What's the depth of each filter?
16

3. What's the output size?
3 x 3 x 36

The convolutional layer - the math behind

- A discrete **convolution between two vectors** with finite size **x** and **w** is mathematically defined as:

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{w} \ \rightarrow \ y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]\, w[k]$$

- **w** is typically called the **filter or kernel**
- The index i runs through each element of the output vector y
- In machine learning applications **we always deal with finite feature vectors**

The convolutional layer - the math behind

- Let's assume that **x** and **w** have **n** and **m** elements respectively, where $\boldsymbol{m} \leq \boldsymbol{n}$:

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{w} \rightarrow y[i] = \sum_{k=0}^{k=m-1} x[i+m-k]w[k]$$

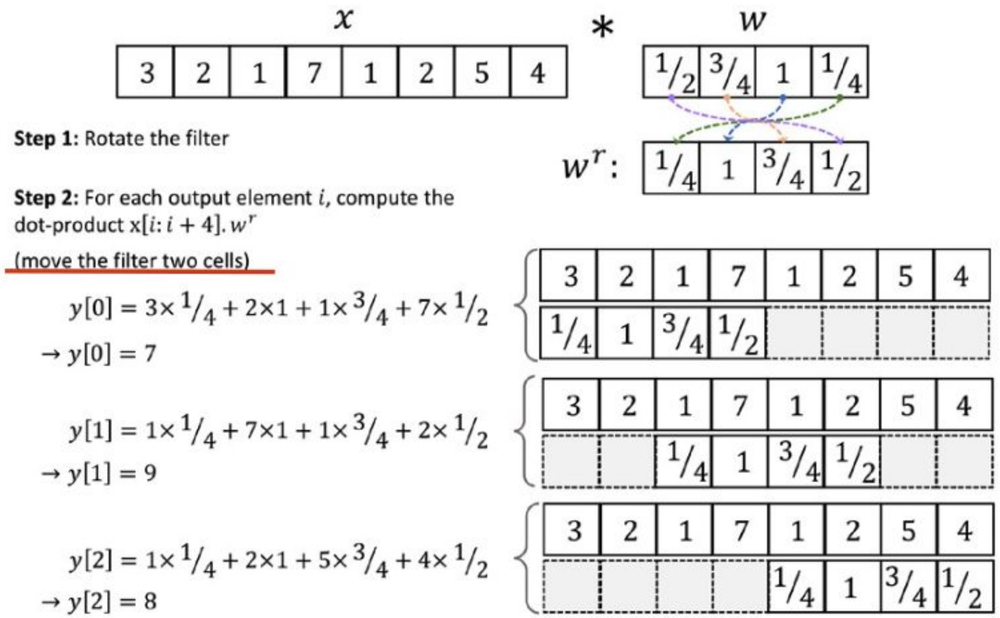
- It's important to notice that **x** and **w** are indexed in different directions in this summation

- Computing the sum with one index going in the reverse direction is equivalent to computing the sum with **both indices in the forward direction after flipping one of those vectors**

- This operation is repeated in a sliding window approach to get all the output elements

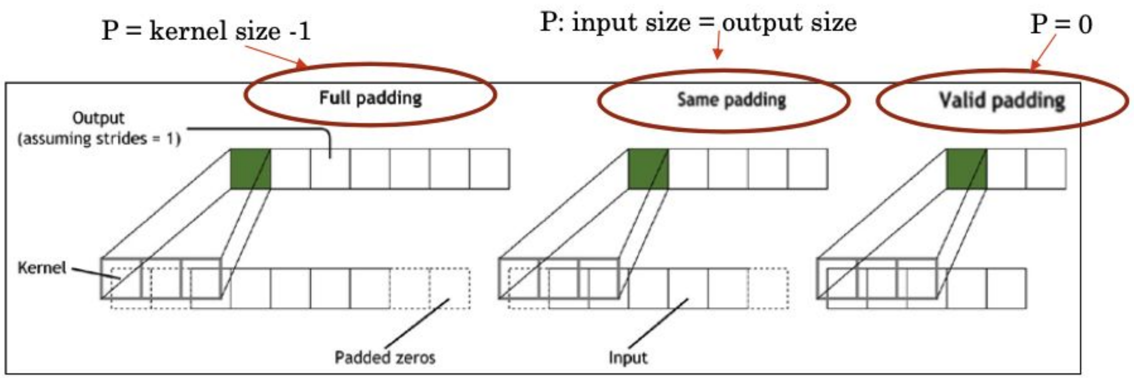
The convolutional layer - a practical example

x = [3 2 1 7 1 2 5 4], w = [½, ¾, 1, ¼]



Padding layer

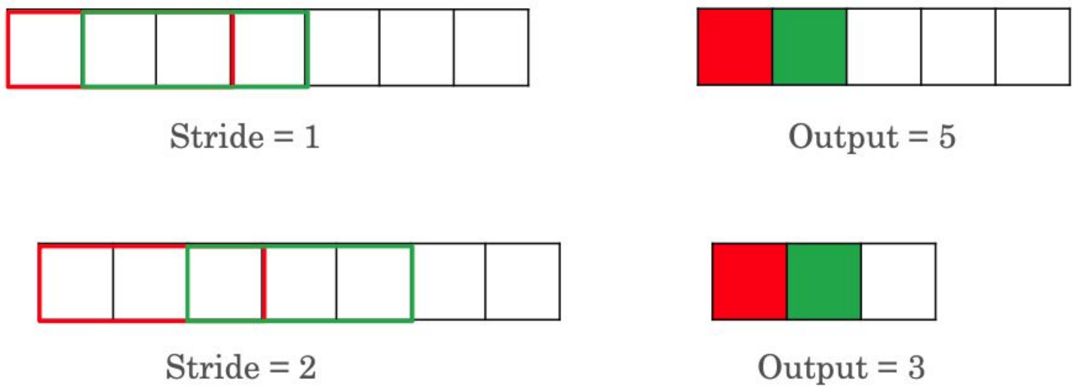
- The result of this convolution is a tensor with a smaller shape than the input one
- To preserve/increase shape a **padding procedure** can be applied
  - It consists in padding zero pixels to input tensor
  - Usually, a **same padding** procedure is used, meaning that the output vector has the same size as the input one.



Moving along the input: Strides

- One concept introduced in the previous example is the **number of cells the filter is moved when shifted** across the vector x (to pass from a y index to another)

- It is called strides
- **Example: N=7, filter=3**



The output size

- The size of the vector obtained by a convolution can be calculated as follows:

$$o = \left[\frac{n + 2p - m}{s}\right] + 1$$

- **o = output dimension**
- n = input dimension
- p = padding
- m = kernel size
- s = stride

Convolution in 2D

- Extend what we said in 2D:

$$\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{W} \;\rightarrow\; Y[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i-k_1, j-k_2]\, W[k_1, k_2]$$

- $\boldsymbol{X}_{n1 \times n2}$ and $\boldsymbol{W}_{m1 \times m2}$ are now two matrices → Y is a **2D matrix** as well

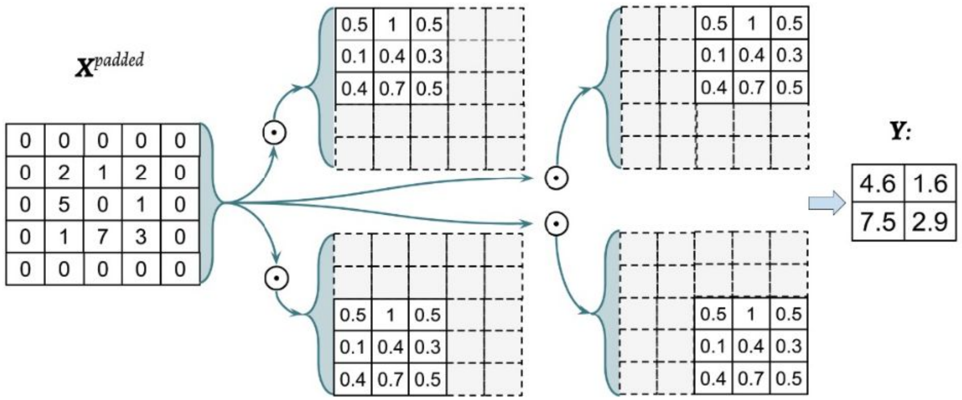
Example:
- input matrix $\boldsymbol{X}_{3\times3}$
- kernel matrix $\boldsymbol{W}_{3\times3}$
- p=(1, 1)
- stride s=(2, 2)

$$\boldsymbol{X}: \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 2 & 0 \\ 0 & 5 & 0 & 1 & 0 \\ 0 & 1 & 7 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \boldsymbol{W}: \begin{bmatrix} 0.5 & 0.7 & 0.4 \\ 0.3 & 0.4 & 0.1 \\ 0.5 & 1 & 0.5 \end{bmatrix}$$

Convolution in 2D

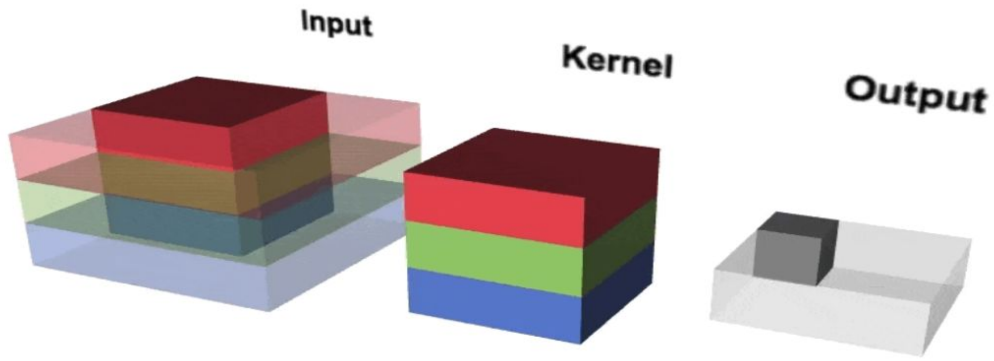
- **Rotate the filter** to perform the sum on indices running in the same directions

$$W^r = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.1 & 0.4 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$



How does a convolutional layer work on a RGB image?

- For each channel color there is a different filter
- The three outputs are added together
- **The output of a convolutional layer with a multi-layer input is a single layer**



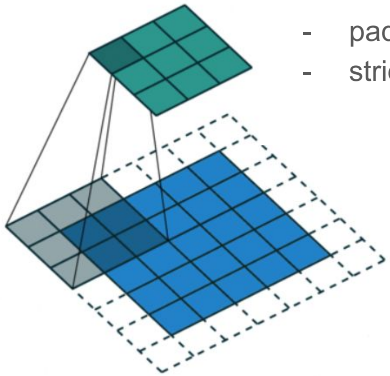
Strides and padding in 2D

- Zero-padding and strides concepts are the same of 1D case
- The output size of a 2D filter is still calculable with the formula seen before, applied on width and height separately

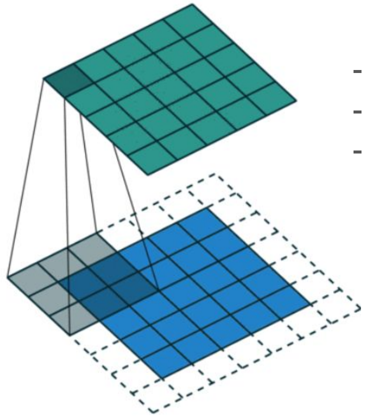
Input dimension Zero-padding Kernel dimension

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

stride



- padding (1,1)
- strides (2,2)

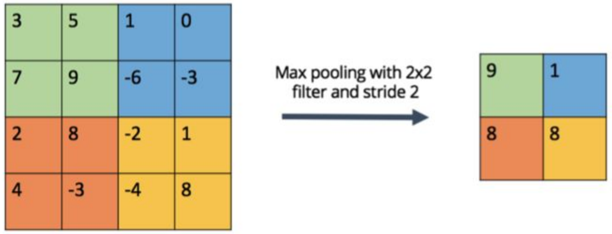


- padding (1,1)
- strides 1
- The input shape is preserved (padding 'same')

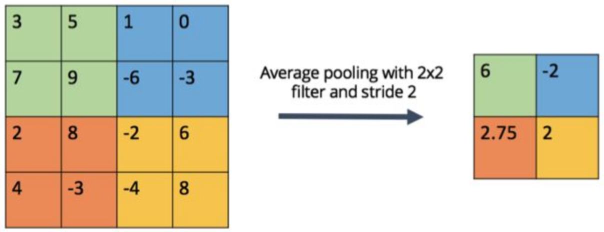
The pooling layer

- According to zero-padding and strides it is possible to change (usually **reduce) the input dimension**
- This task can be also performed with a **"Pooling" layer**

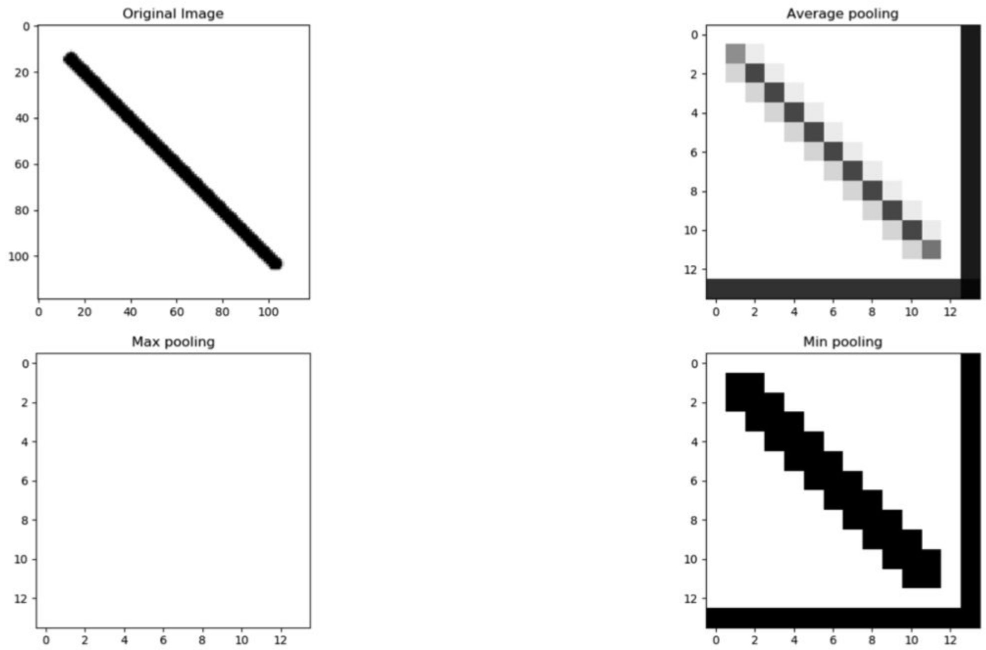
- Two kinds of pooling:
  - **Maximum Pooling** (or Max Pooling): Calculate the maximum value for each patch of the feature map.



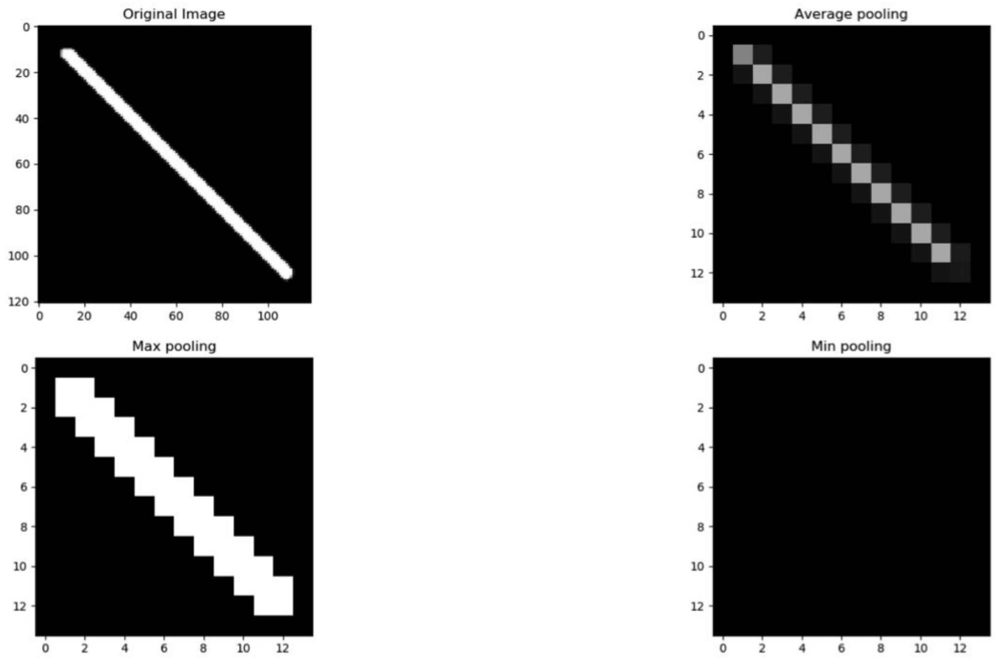
  - **Average Pooling:** Calculate the average value for each patch on the feature map.



The effect of the pooling layer



The effect of the pooling layer



Max Pooling

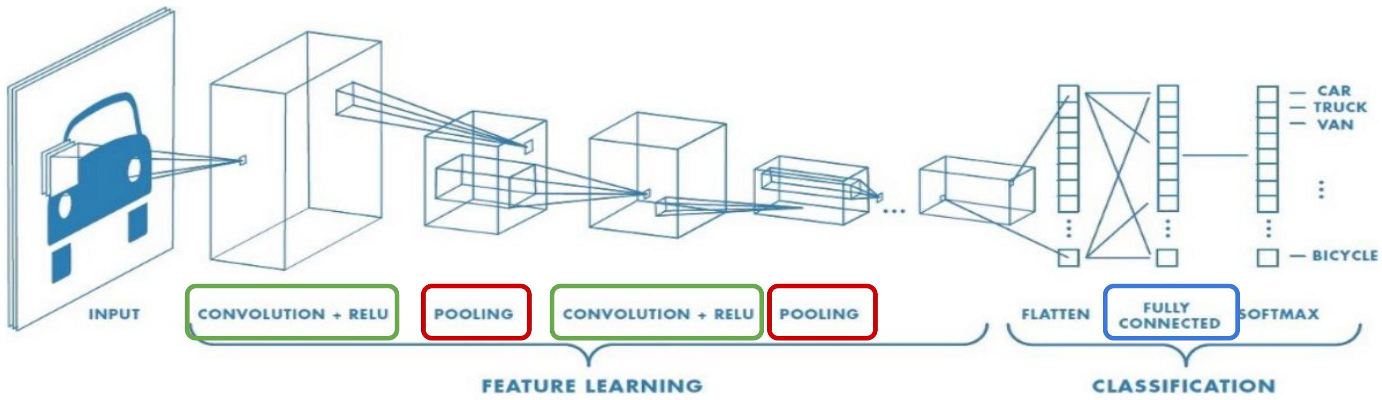
- It basically introduces a **local invariance**
  - **Small changes** in a local neighbourhood **do not change the result of max-pooling**

$$
\left.\begin{matrix}
\boldsymbol{X}_1 = \begin{bmatrix} 10 & 255 & 125 & 0 & 170 & 100 \\ 70 & 255 & 105 & 25 & 25 & 70 \\ 255 & 0 & 150 & 0 & 10 & 10 \\ 0 & 255 & 10 & 10 & 150 & 20 \\ 70 & 15 & 200 & 100 & 95 & 0 \\ 35 & 25 & 100 & 20 & 0 & 60 \end{bmatrix} \\
\\
\boldsymbol{X}_2 = \begin{bmatrix} 100 & 100 & 100 & 50 & 100 & 50 \\ 95 & 255 & 100 & 125 & 125 & 170 \\ 80 & 40 & 10 & 10 & 125 & 150 \\ 255 & 30 & 150 & 20 & 120 & 125 \\ 30 & 30 & 150 & 100 & 70 & 70 \\ 70 & 30 & 100 & 200 & 70 & 95 \end{bmatrix}
\end{matrix}\right\} \xrightarrow{\text{max pooling } P_{2\times 2}} \begin{bmatrix} 255 & 125 & 170 \\ 255 & 150 & 150 \\ 70 & 200 & 95 \end{bmatrix}
$$

- **Why we use it?**
  - Pooling decreases the size of features → **higher efficiency**
  - Reduces the risk of overfitting

- Typically, pooling is assumed to be non-overlapping (pooling size = stride)

The big picture

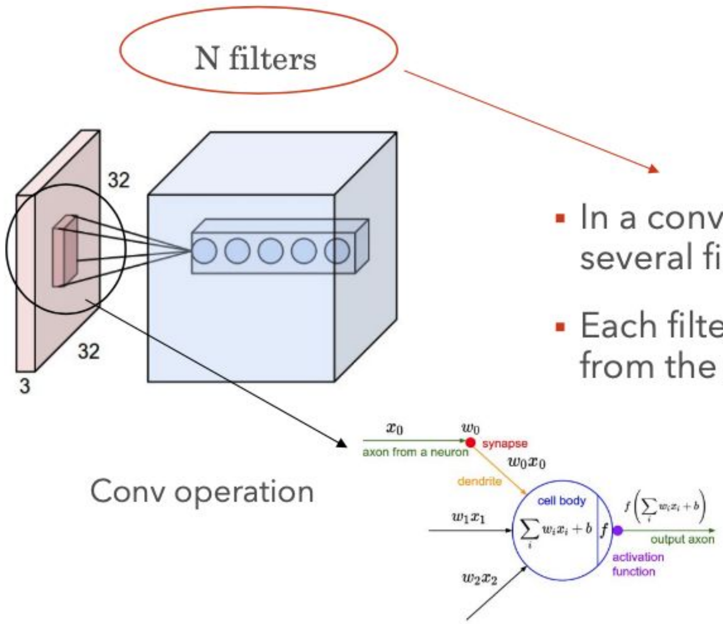


Take away messages

- FILTERS WEIGHTS ARE **LEARNT FROM DATA DURING TRAINING**
- THE NETWORK LEARNS WHICH ARE THE **MOST DISCRIMINANT PATTERNS**
- A CNN PERFORMS THE **CLASSIFICATION** BY READING THESE **EXTRACTED FEATURES**
- ON THE CONTRARY, A **DNN** READS ONLY **PIXELS VALUES**

CNN recap

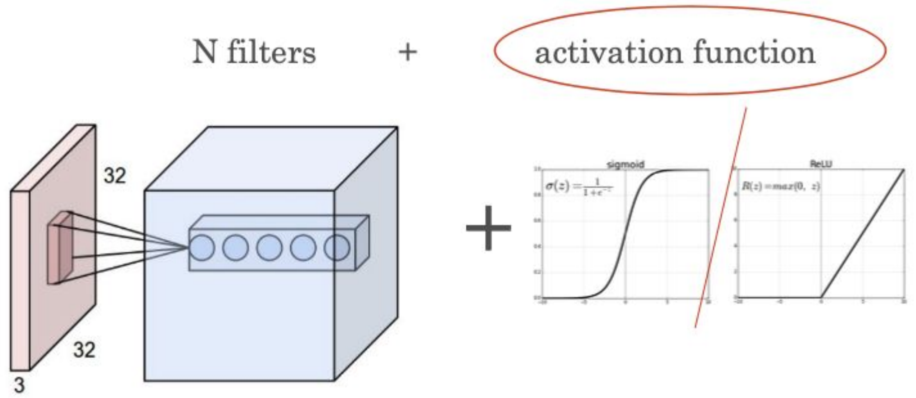
- Typically, a convolutional layer is composed of:



- In a convolution layer, usually, not only one but several filters are stacked together
- Each filter learns some different information from the same image

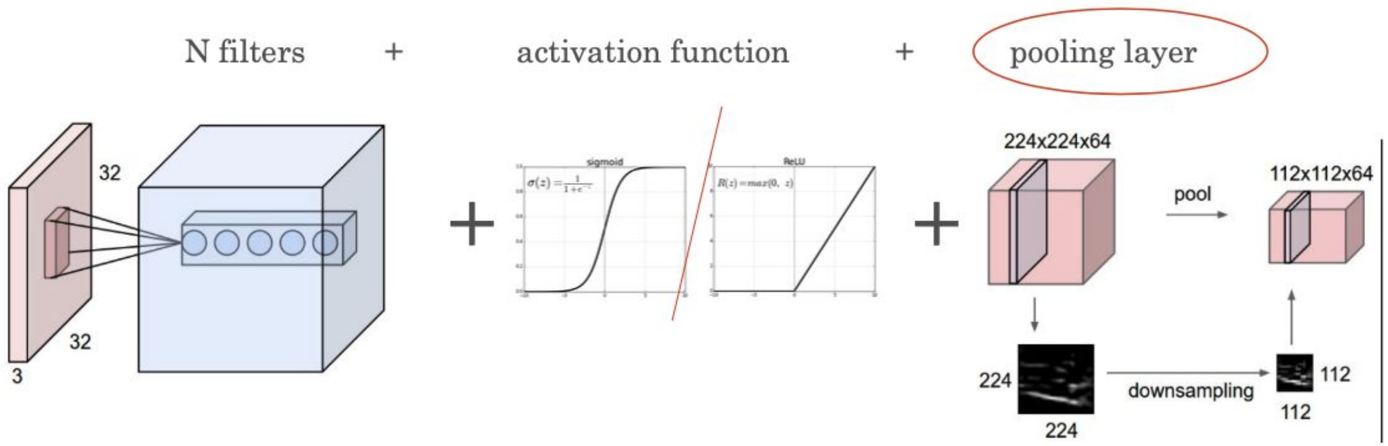
CNN recap

- Typically, a convolutional layer is composed of:



CNN recap

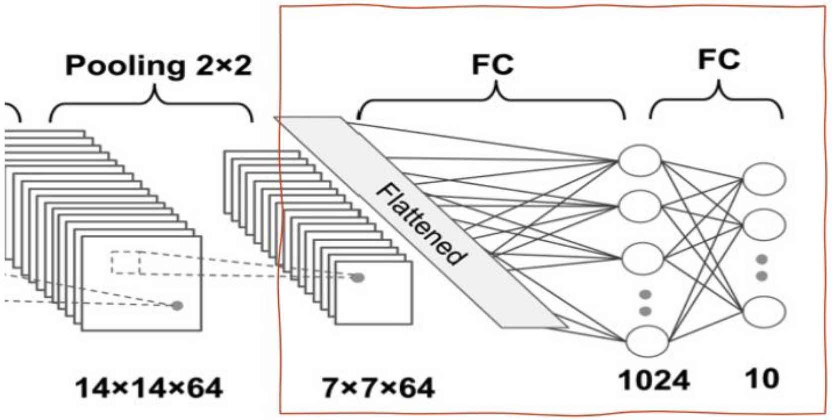
- Typically, a convolutional layer is composed of:



Dense layers for classification

- Now we must **flatten the final output** and feed it to a regular FFNN for classification purposes
- Adding a Fully-Connected layer is a way of **learning non-linear combinations of the high-level features** (from filters)

    - The image is flattened into a column vector
    - Then fed to a fully-connected neural network



Different loss functions for different classification problems

- Depending on the type of problem and the type of output, we should choose the **appropriate loss function to train our model**

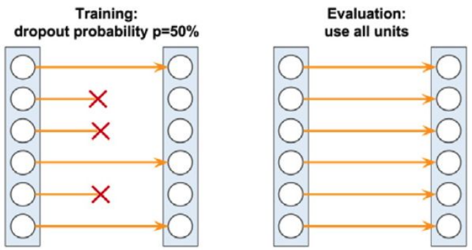
Loss function	Usage	Examples	
		Using probabilities `from_logits=False`	Using logits `from_logits=True`
`BinaryCrossentropy`	Binary classification	y_true: 1 y_pred: 0.69	y_true: 1 y_pred: 0.8
`CategoricalCrossentropy`	Multiclass classification	y_true: 0 0 1 y_pred: 0.30 0.15 0.55	y_true: 0 0 1 y_pred: 1.5 0.8 2.1
`Sparse CategoricalCrossentropy`	Multiclass classification	y_true: 2 y_pred: 0.30 0.15 0.55	y_true: 2 y_pred: 1.5 0.8 2.1

- **Logit**: sometimes preferred due to numerical stability reasons

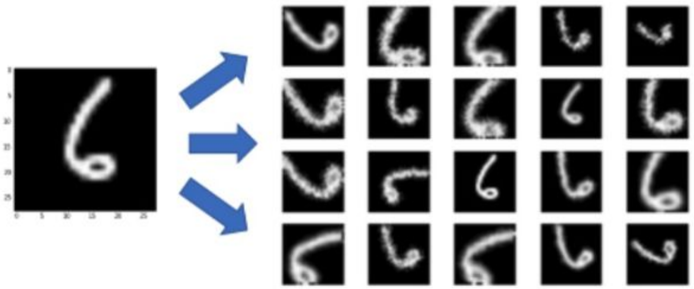
$$\operatorname{logit}(p) = \ln\left(\frac{p}{1-p}\right)$$

DNN regularization argument apply also for CNN

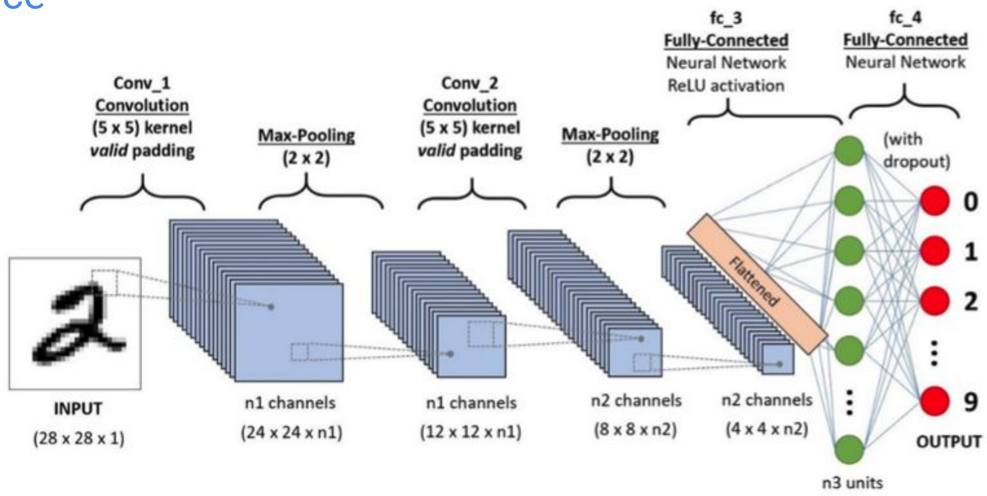
- **Dropout:** to avoid overfitting



- **Data augmentation:** invariance to translation



Good practice



- Since **the first convolutional filters** learn high level features in the image in input and the input size is larger than in inner layers, the **number of filters is relatively small** to not insert too many weights
- A good practice is **to increment this number in the subsequent convolutional steps**
- **Dropout can be inserted not only between dense layers but also between a Conv layer and its input**

An example:
Cats vs Dogs

Hands on example

- A dataset containing photos of dogs and cats is provided
  - Pre-processing the images
  - Creating a model
  - Training and testing
  - Evaluating performances
  - Improving the model!
- Have a look [here]!

Your turn

Aerial images

- Scene classification with 21 categories
- Images available here