SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

Unione Parmense
degli Industriali

GIA
Gruppo Imprese

Gruppo Imprese
Artigiane

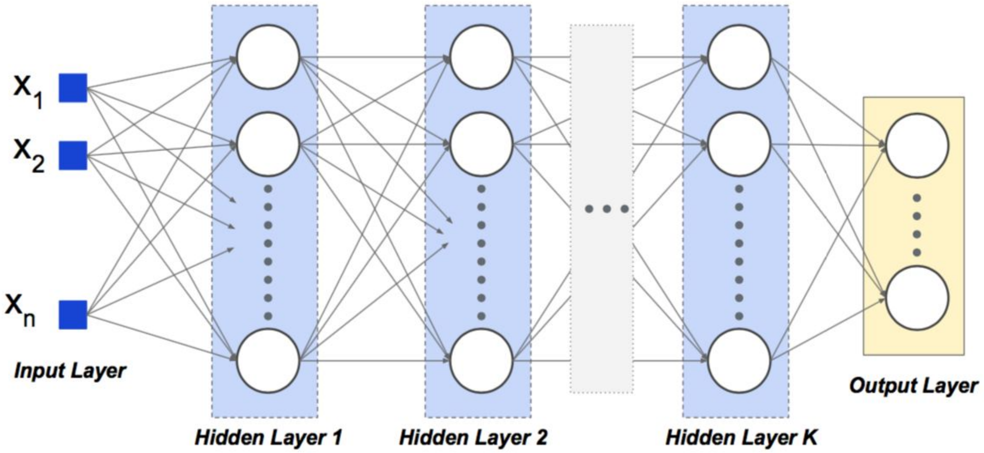
Convolutional Neural Networks

Artificial Intelligence and Machine Learning

Gianfranco Lombardo MEng, Ph.D Candidate in ICT
gianfranco.lombardo@unipr.it

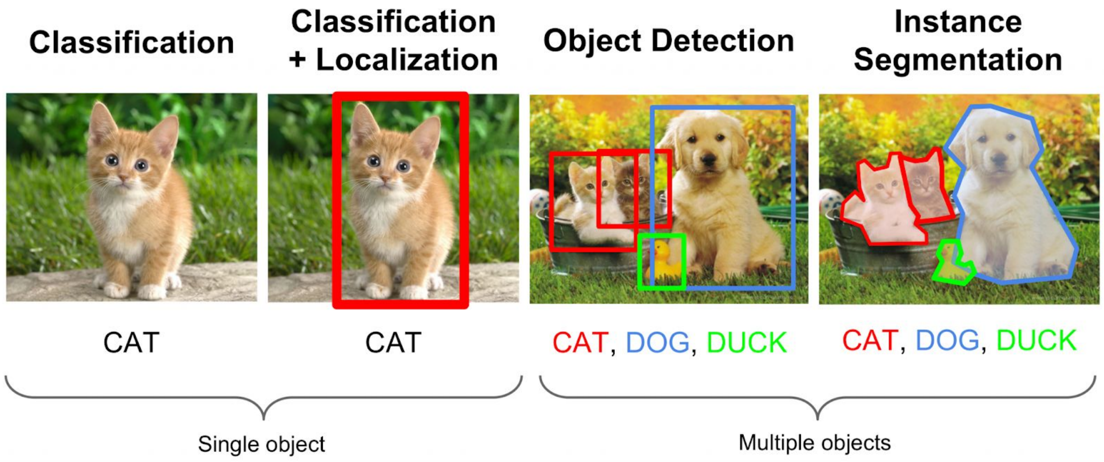
Summary on Neural Networks

- Multilayer Perceptron (MLP)
- Functions approximation
- Generic classification tasks
- In the past used also for more complex tasks with several limits in the training phase

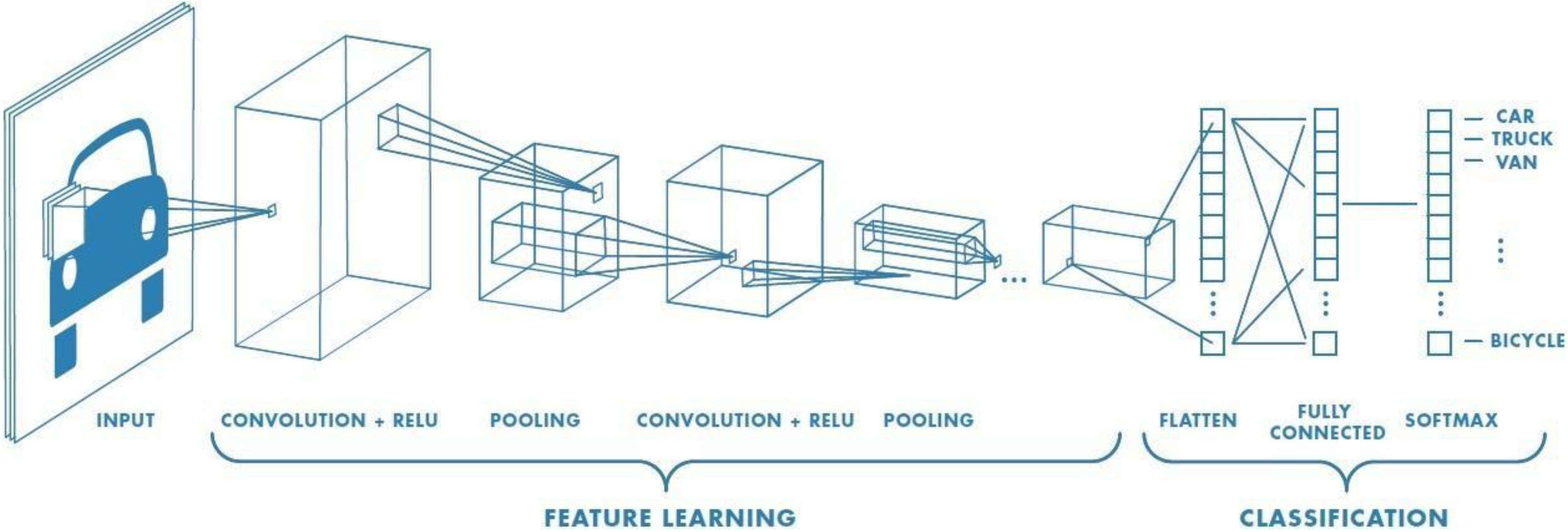


What about image classification and related tasks ?

- Multi-layer perceptron has been used also for image analysis but with several limits:
  - Memory constraints
  - Accuracy results often lower or equal to other less complex models (e.g Support Vector Machine)
  - With high resolution images training is often computationally infeasible



Solution: Convolutional Neural Network (CNN)

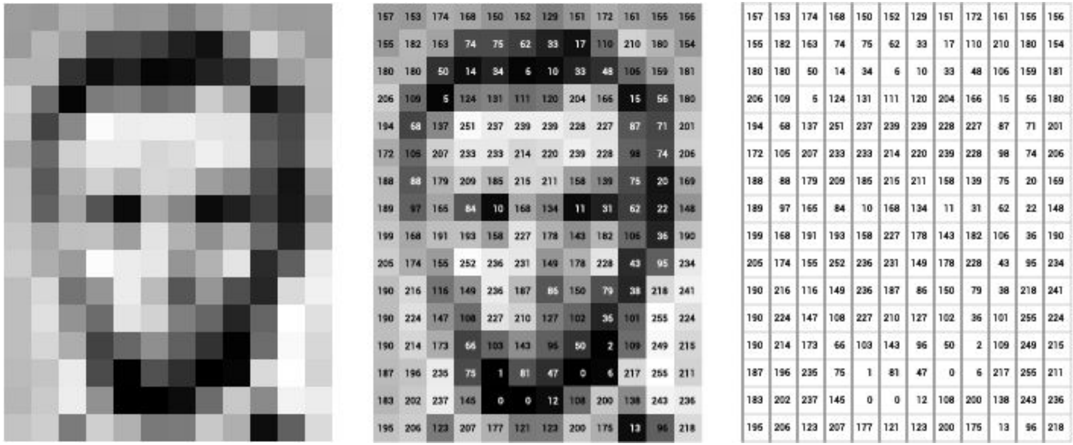


What is a digital image ?

- Grey-scale images can be viewed as a matrix of values (pixels)

- Each pixel has a value between 0 (black) and 255 (white)

- With the lack of colors we can say that these images are “single channel”

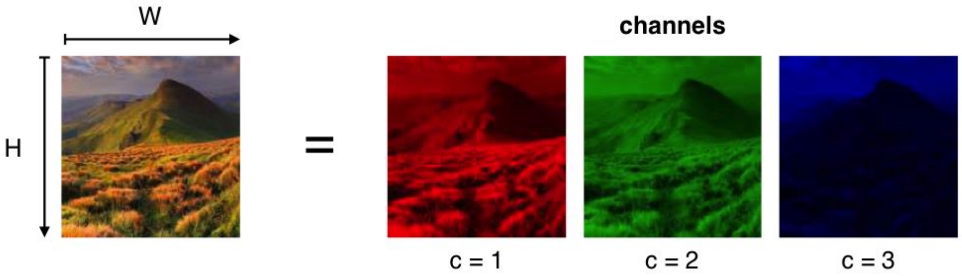


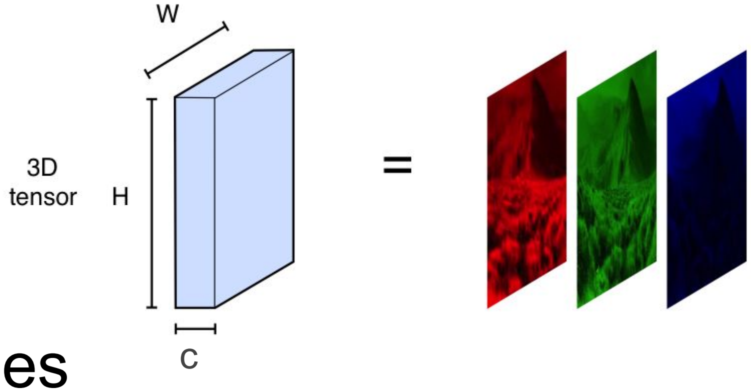
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

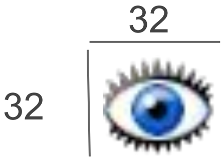
What is a digital image ?

- To have a color image we need more channels to add these information

- For example RGB images can be viewed as a tensor of values:
  - Each layer is a channel describing a color component of the image
  - Dimension are **Width X Height X N ° of Channels**
  - **Example:**







= 32x32x3 = 3072 values

Image classification with a Neural network

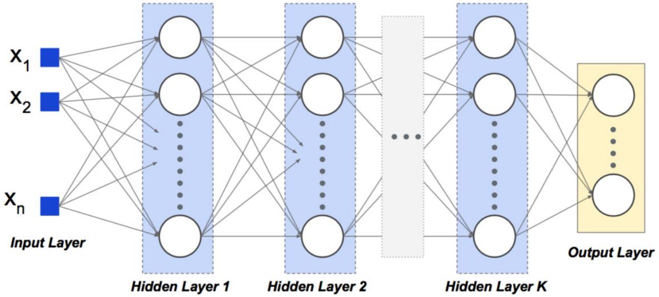
- The basic idea is “to flatten” the image matrix and using each value as an input x_i of the network
- Advantages: No preliminary feature extraction needed
  - Network learns its own feature analyzing the training set at each epoch

h = 720

w = 420

1	...	3
200	...	123
...	...	...
14	...	100

x_1	x_2		x_i		x_n
1	...	..	3	...	100



Let’s recap the basic concepts of a neuron

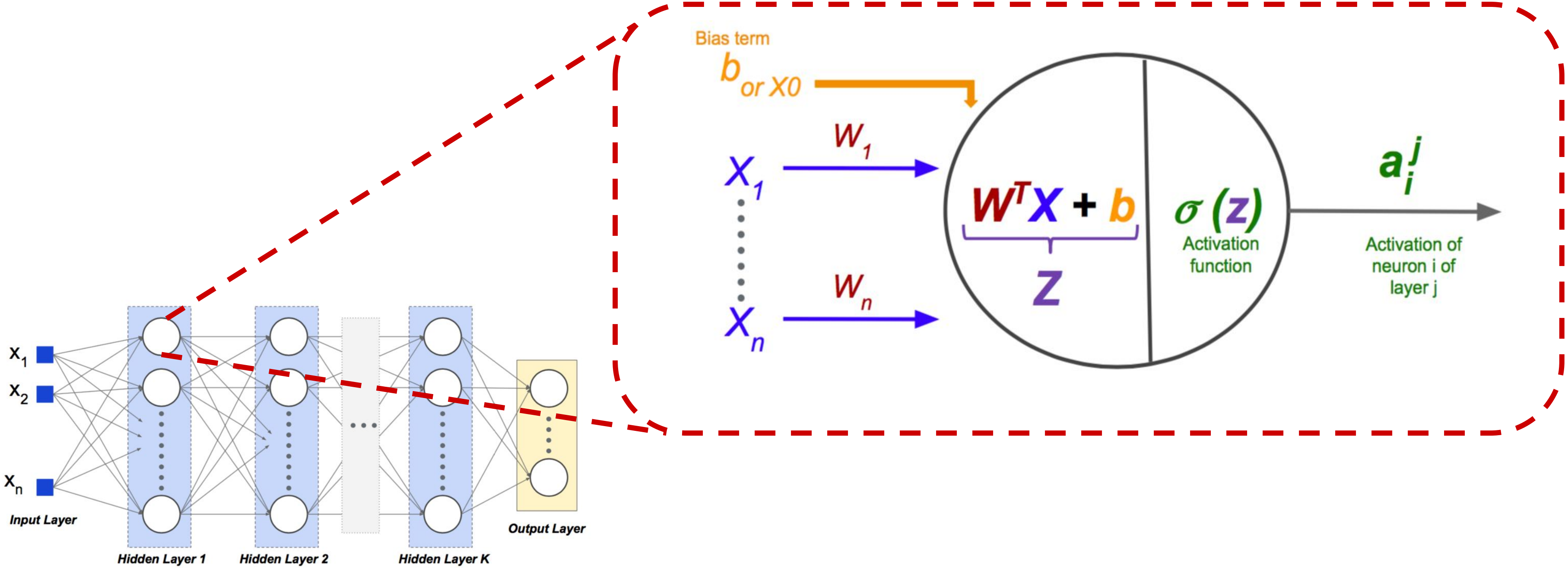
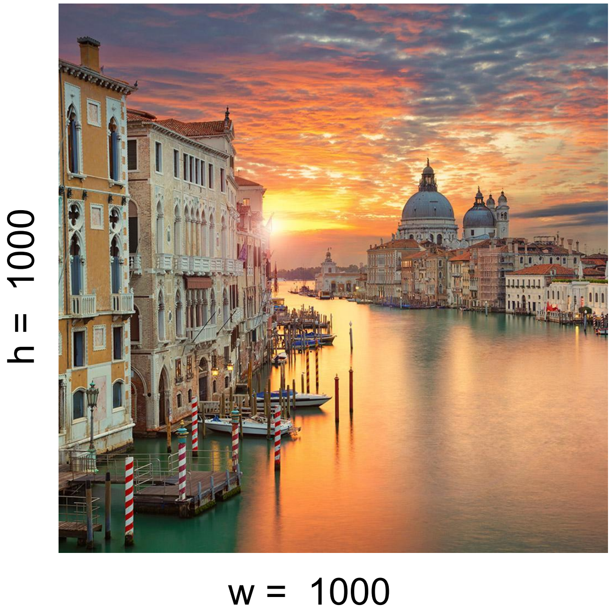


Image classification with a Neural network



- Images with a medium-low resolution of 1 megapixel
- The input matrix shape is 1000x1000x3
- X input matrix contains 3 millions of elements
  - Just consider the number of neurons in the first hidden layer
  - We need many neurons to increase network learning capacity but there are some limits
    - Being J the number of neurons in layer 1
    - The weight matrix W of layer 1 has shape (J, $3x10^6$)
    - If J=1000 -> W has 3 billions of elements
    - If J=100 -> W has 300 millions of elements

Find a tradeoff !

- Using traditional Neural networks with high resolution and color images brings to:
  - ***Too many elements in weight matrix*** for each layer -> ***Difficult avoiding overfitting*** while learning W_{ij} elements with backpropagation algorithm if we do not have a HUGE number of images
  - ***Memory requirements*** can become hard to be satisfied !

Possible tradeoffs:

- Use grey-scale images: *Only 1 channel **VS** loss of information*
- Use low resolution images: Possible in some contexts, *reducing W matrix shape **VS** loss of information*
- Increase hardware capacity: *computational power **VS** costs*

- Used in some real applications in 90s (e.g handwritten postcode recognition etc..)
  - Due to these limitations neural networks based systems have been abandoned for less complex models until about our days with the advent of Deep Learning !!

Not only computational limits !

Main advantage: Automatic feature extraction

- Visual features get extracted in an automatic way by the Neural Network in the learning phase of the inner weights
- No need of important pre-processing steps
- Difficult to understand which features have been considered by the network !
- To increase network capacity in extracting features we need several hidden layers and neurons

PROBLEM: Loose of local Information !!!

- To fed the neural network we have to flatten the image, losing its 2D structure
- It can affect results in an important way in some kinds of classification tasks !

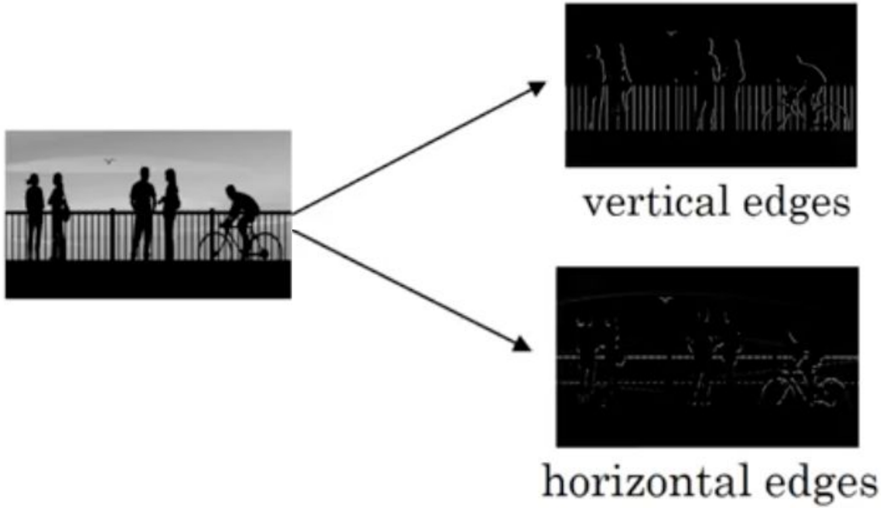
⬇

- In fact, in traditional Computer Vision models, visual features get extracted by using **Kernels (or filters)** and the **convolutional operator** on images!

Filters and convolution operator on images

- Manual extraction of visual features on images is usually conducted by convolving filters (or kernels) on images

- Each desidered feature can be extracted by convolving the image with the appropriate filter



Filters and Convolution operator

- Convolution is the process of adding each element of the image to its local neighbors, weighted by the filter

- The resulting image is obviously compressed in its dimension

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Image

Convolution operator

*****

Filter/Kernel

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

=

Convolved features

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

$$(3*1+0*0+1*-1) + (1*1+5*0+8*-1)+(2*1+7*0+2*-1) = -5$$

Filters and Convolution operator

- Convolution is the process of adding each element of the image to its local neighbors, weighted by the filter

- The resulting image is obviously compressed in its dimension

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Image

Convolution operator

*****

Filter/Kernel

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

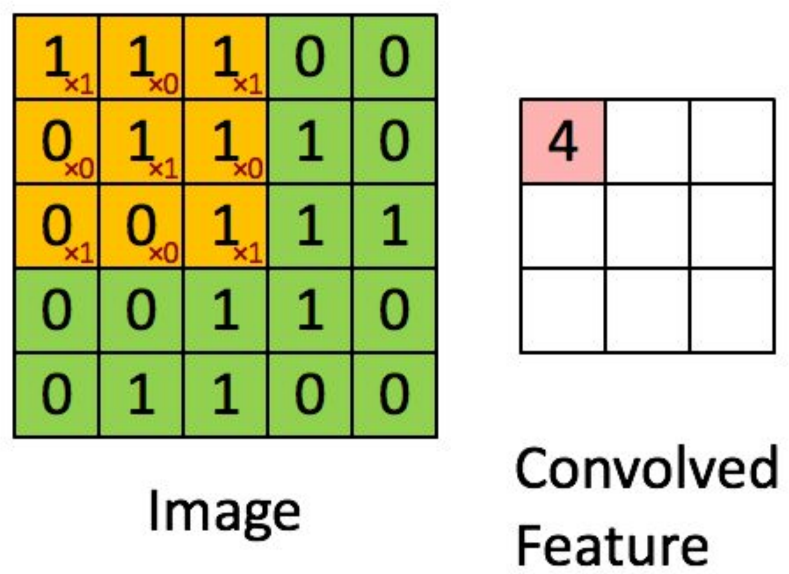
=

Convolved features

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

(3*1+0*0+1*-1) + (1*1+5*0+8*-1)+(2*1+7*0+2*-1) = -5

Dimension of convolved feature ?



- Image has dimension mxn
- Filter has dimension fxf
- The resulting convolved feature has dimension:

  (m-f+1) x (n-f+1)

Example: Vertical edges detection

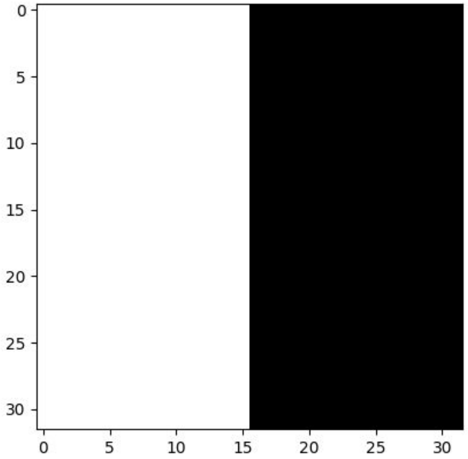


Image
32x32

Convolution operator

*

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

Filter 3x3 for vertical edges

=

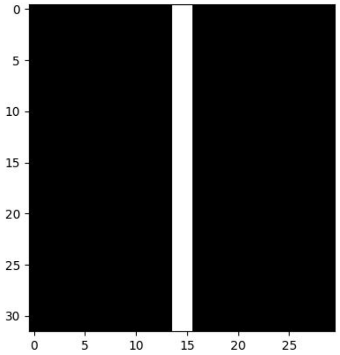
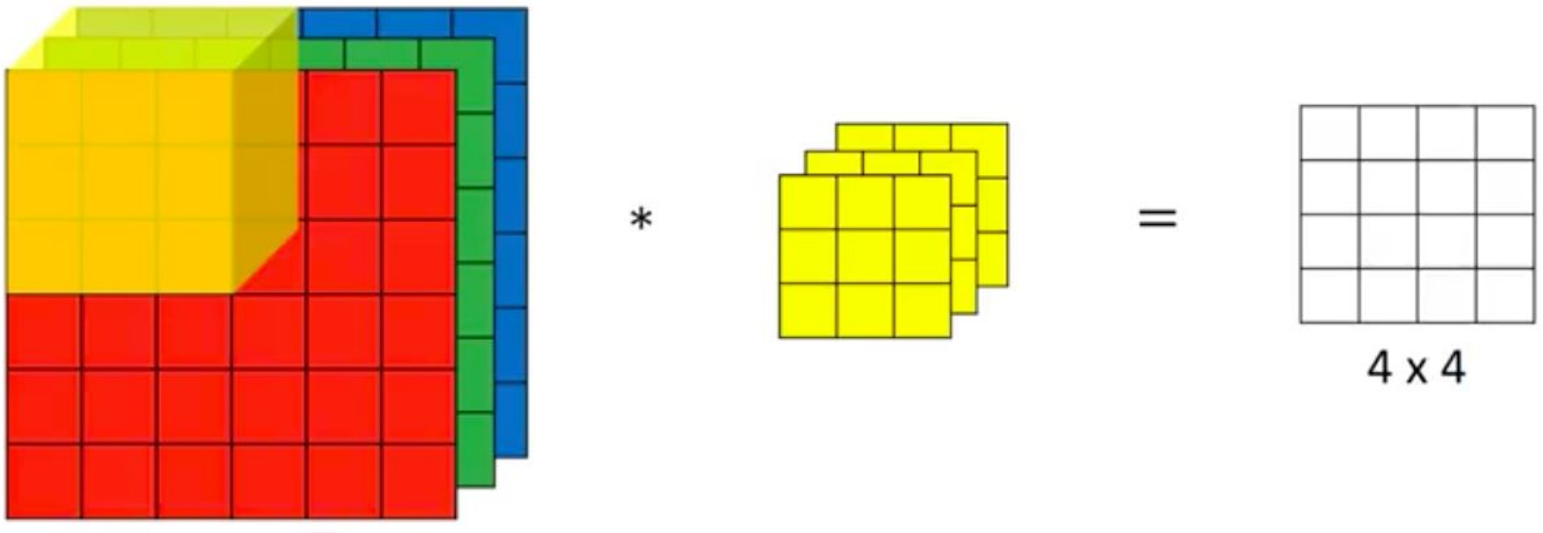


Image
30x30

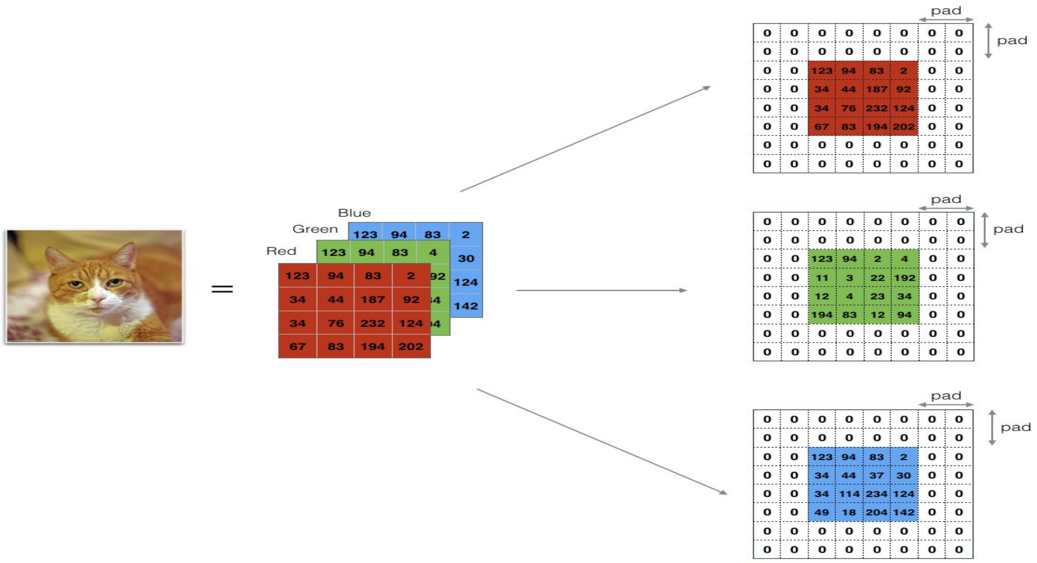
Convolution over volumes (RGB image)



Note: Filter’s channel number has to be equal to the number of image channels !

Avoid the image shrinking output: Padding !

- Everytime we perform a convolution, we shrink the output image
- To avoid it, it is useful to pad the input image with other values at the border
- It helps also to keep more of the information at the border of the input image



123	94	83	2
34	44	187	92
34	76	232	124
67	83	194	202

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	123	94	83	2	0	0
0	0	34	44	187	92	0	0
0	0	34	76	232	124	0	0
0	0	67	83	194	202	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	123	94	2	4	0	0
0	0	11	3	22	192	0	0
0	0	12	4	23	34	0	0
0	0	194	83	12	94	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	123	94	83	2	0	0
0	0	34	44	37	30	0	0
0	0	34	114	234	124	0	0
0	0	49	18	204	142	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Strided convolution

- A variant of the classic convolution operator that consists in making some “jumps” convolving the filter on the image
- It is like a shallow convolution that can permit us to increase or manage the shrinking of the output

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7x7 image

*

Filter

3	4	4
1	0	2
-1	0	3

Stride S = 2

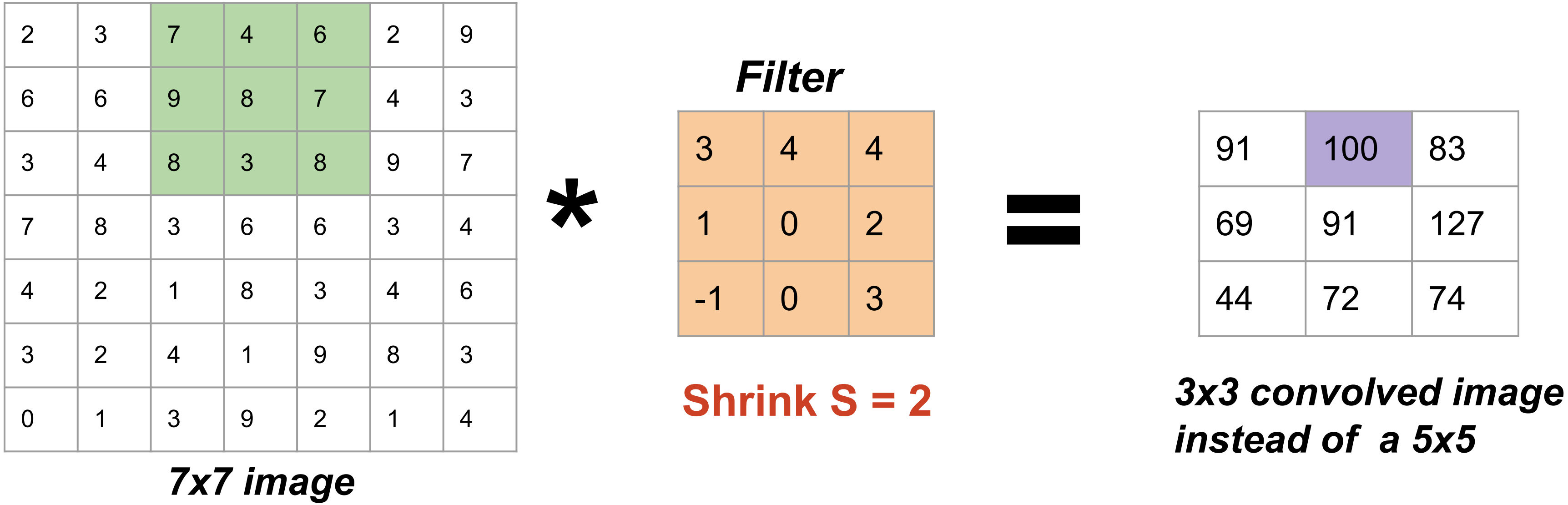
=

91	100	83
69	91	127
44	72	74

3x3 convolved image instead of a 5x5

Strided convolution

- A variant of the classic convolution operator that consists in making some “jumps” convolving the filter on the image
- It is like a shallow convolution that can permit us to increase or manage the shrinking of the output



2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7x7 image

*

Filter

3	4	4
1	0	2
-1	0	3

Shrink S = 2

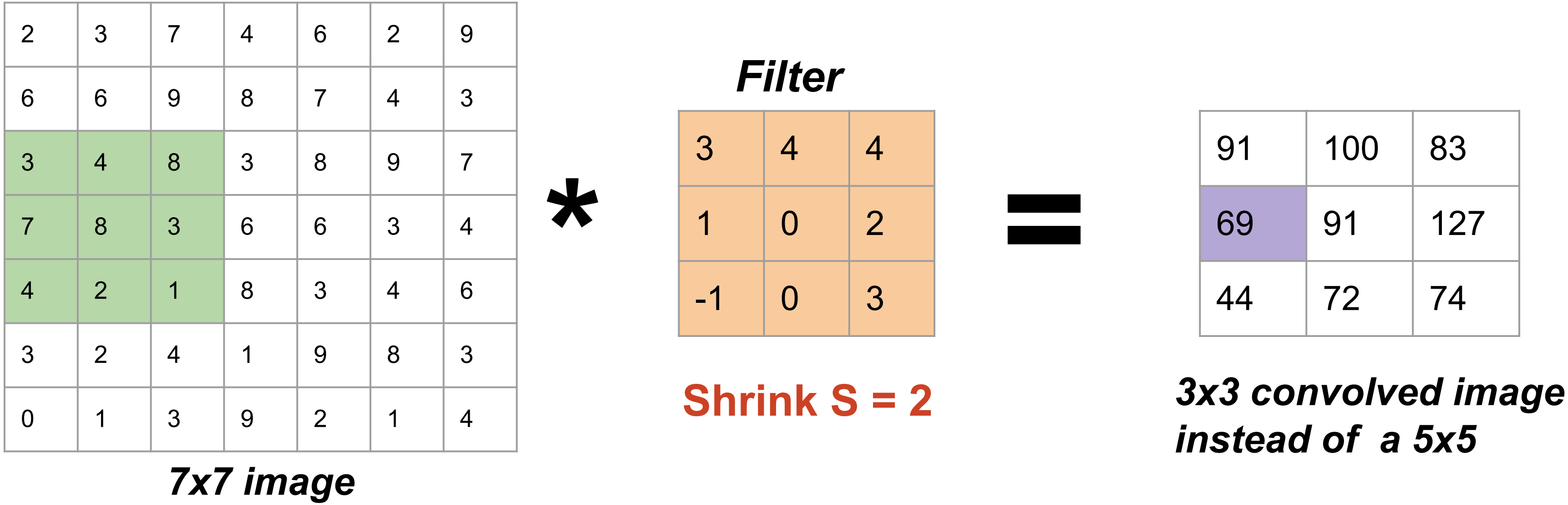
=

91	100	83
69	91	127
44	72	74

3x3 convolved image instead of a 5x5

Strided convolution

- A variant of the classic convolution operator that consists in making some “jumps” convolving the filter on the image
- It is like a shallow convolution that can permit us to increase or manage the shrinking of the output



2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7x7 image

*

Filter

3	4	4
1	0	2
-1	0	3

Shrink S = 2

=

91	100	83
69	91	127
44	72	74

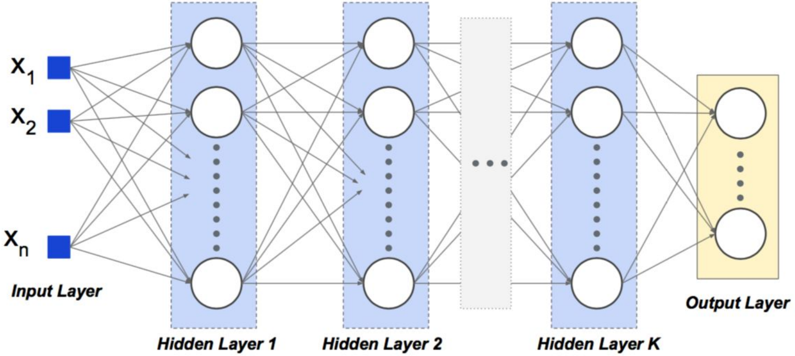
3x3 convolved image instead of a 5x5

How to calculate the final dimension of the output image

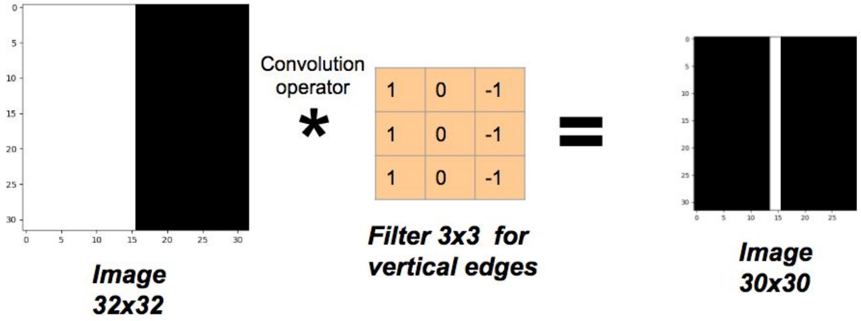
$$\left\lfloor \frac{m+2p-f}{s} + 1 \right\rfloor, \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor, Nf$$

- m : Width of the image
- n: Height of the image
- f : Dimension of the fxf filter
- p : Padding pixels
- s: Striding pixels
- Nf: Number of filters

So...let’s recap



- Automatic feature extraction while learning network weights
- Flattening the image causes the loss of local information
- Explosion of the computation complexity due to the flatten image



$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

- Once features have been extracted convolving the appropriate filters on the images, we can use those features to train an ad-hoc machine learning model with different types of algorithms
  - The choose of the algorithm depends on the target task
- **The choice of the appropriate filters is not so easy, we have to refer to literature or try to invent our own filters !**

Why do not join these two approaches ?

- The perfect tradeoff would be:
  - Automatic feature extraction as in the Neural network model
  - Using kernels approach to reduce W matrix shapes in each layer and without losing local information in the images
  - Learn automatically the appropriate filters as weights of the network
  - Consequently, carry out an automatic feature extraction based on kernels approach

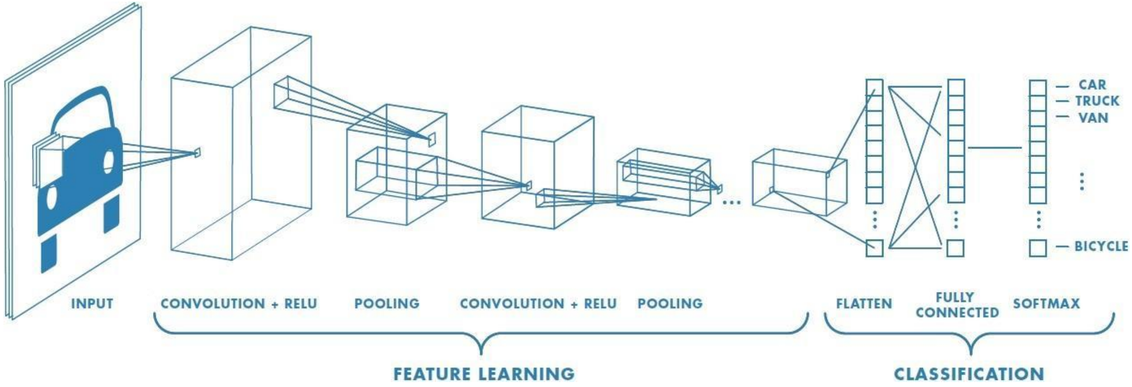
Convolutional Neural Networks !!

Solution: Convolutional Neural Network (CNN)

- A Deep learning architecture proposed by Yann LeCun in 90s but only today it is trainable in an efficient way and with several layers thanks to GPU's

- It is bio-inspired, because the organization of the neurons resembles the organization of the animal visual cortex

- Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

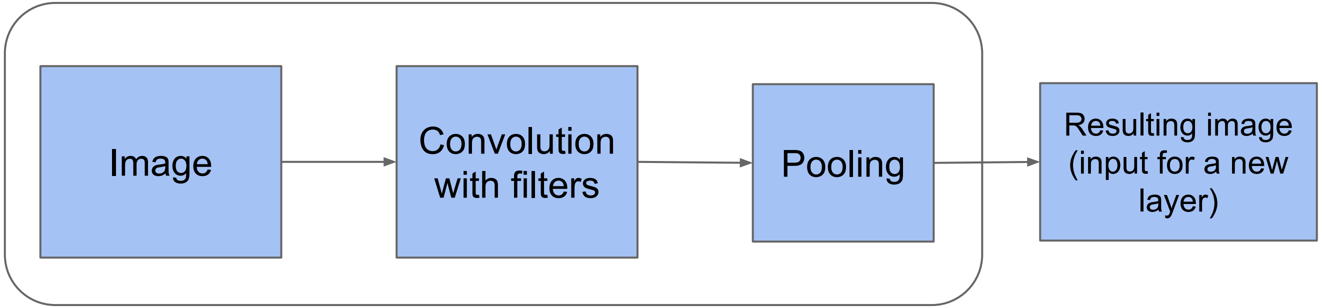


The convolutional layer

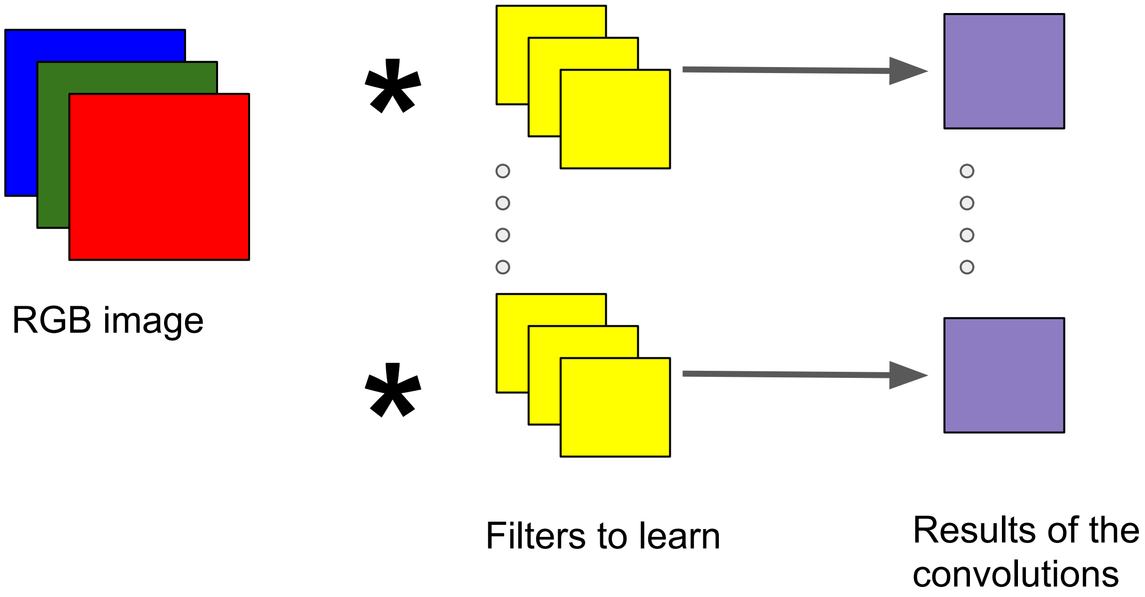
- In the convolutional layer, neurons are arranged in a 2D structure

- It is composed by two operators:
  - Convolution operator
  - Pooling operator

- Each layer has as input a tensor with shape (Width, Height, channels)



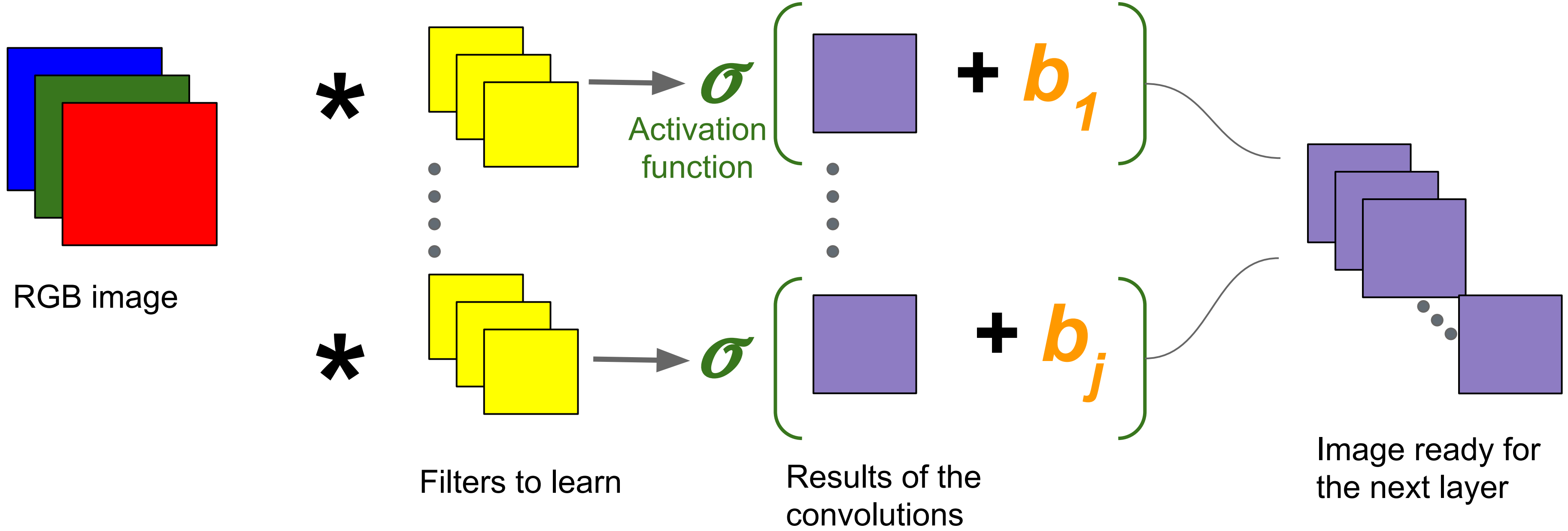
Convolutional operator in the Convolutional layer



- Weights to learn are the elements of filters tensors
- Let’s call the result of the interaction between input data and weights Z = Input*Weights

- We also need to apply an activation function to Z

Convolutional operator in the Convolutional layer



Shapes in the Convolutional Layer

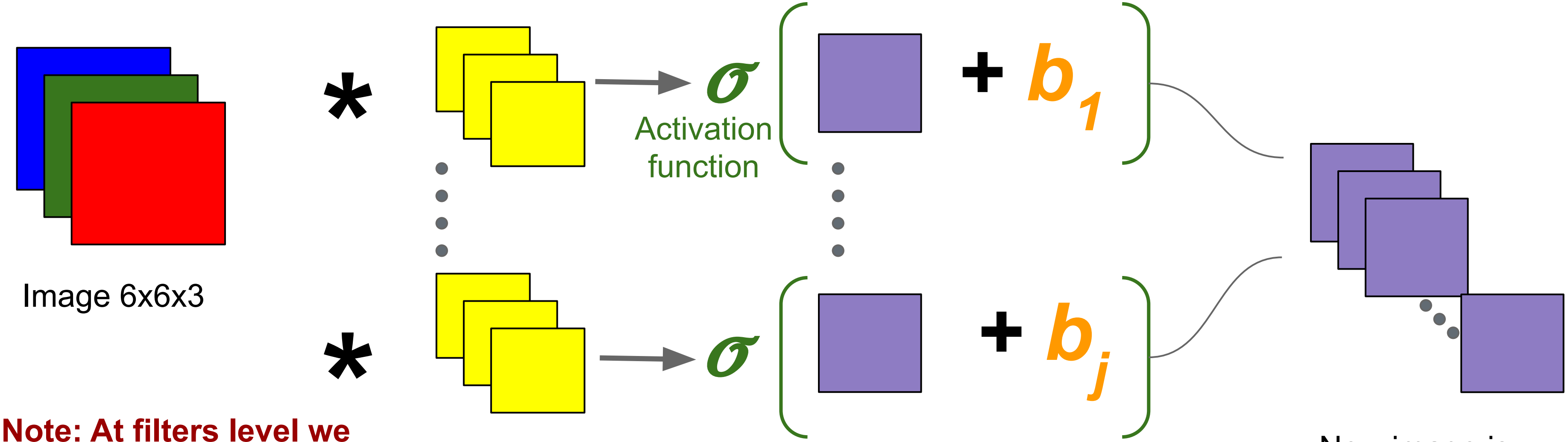


Image 6x6x3

Activation function

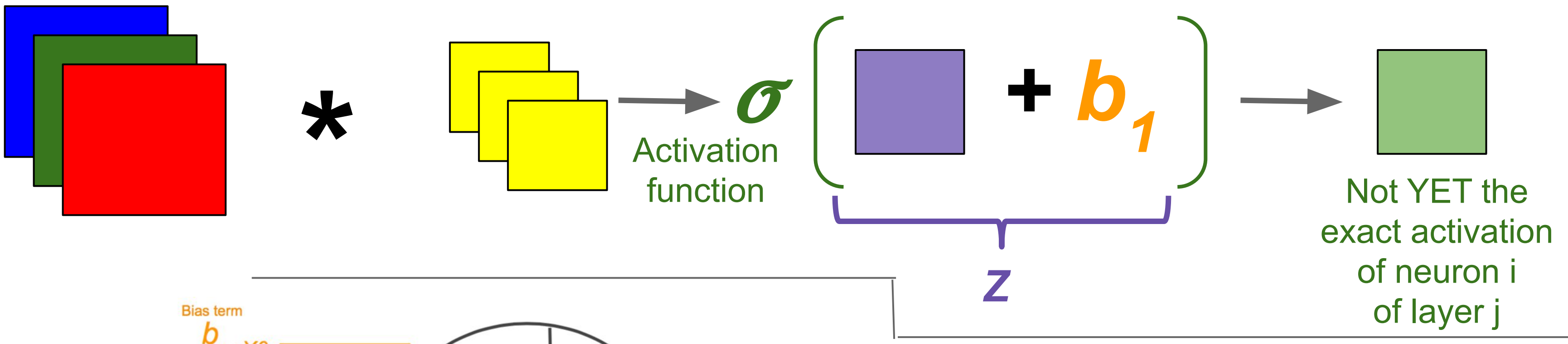
10 Filters 3x3x3

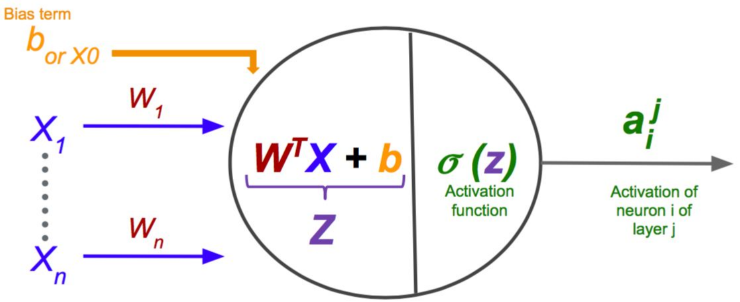
Results of each convolution is 4x4

New image is 4x4x10

Note: At filters level we have 28*10 parameters independent from the image shape !!!

Comparison between Neurons and input data

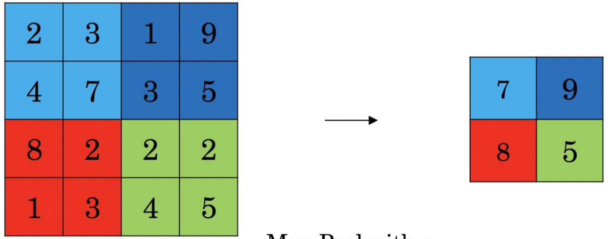




The pooling operator

- Last step of the Convolutional layer
- It doesn’t learn any parameters but makes network computation faster by reducing the dimension of the representations
- It is also useful for the network to summarize the feature learned by each neuron to make these features more robust

Max Pool



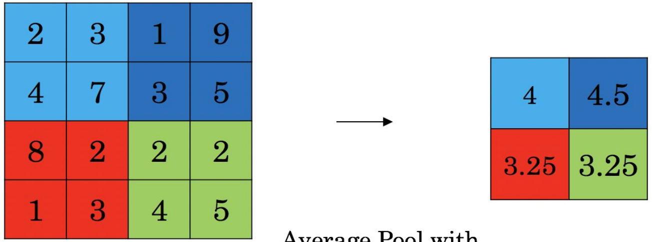
2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

→

7	9
8	5

Max-Pool with a 2 by 2 filter and stride 2.

Average Pool



2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

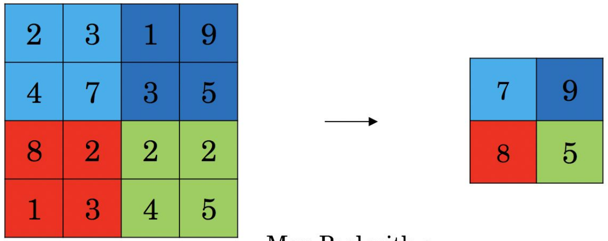
→

4	4.5
3.25	3.25

Average Pool with a 2 by 2 filter and stride 2.

The pooling operators

Max Pool



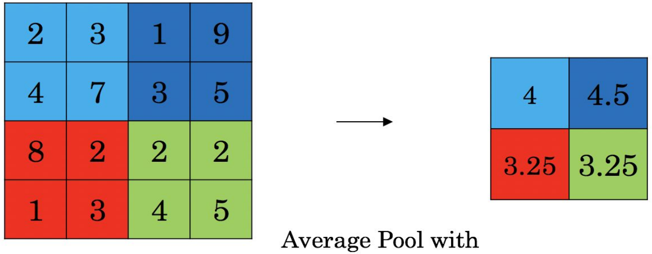
2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

7	9
8	5

Max-Pool with a
2 by 2 filter and
stride 2.

- Most used
- Max-pooling layer: slides an (f,f) window over the input and stores the max value of the window in the output.

Average Pool



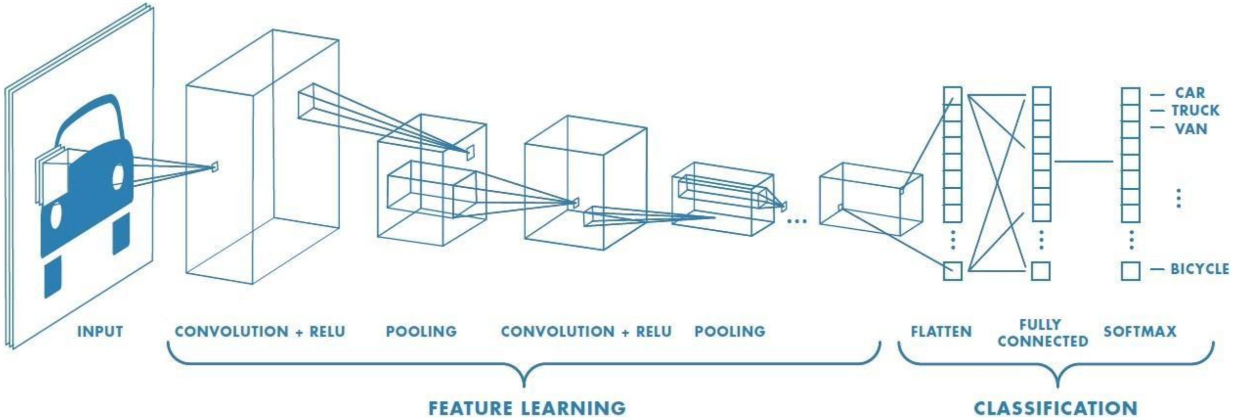
2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

4	4.5
3.25	3.25

Average Pool with
a 2 by 2 filter and
stride 2.

- Average-pooling layer: slides an (f,f) window over the input and stores the average value of the window in the output.

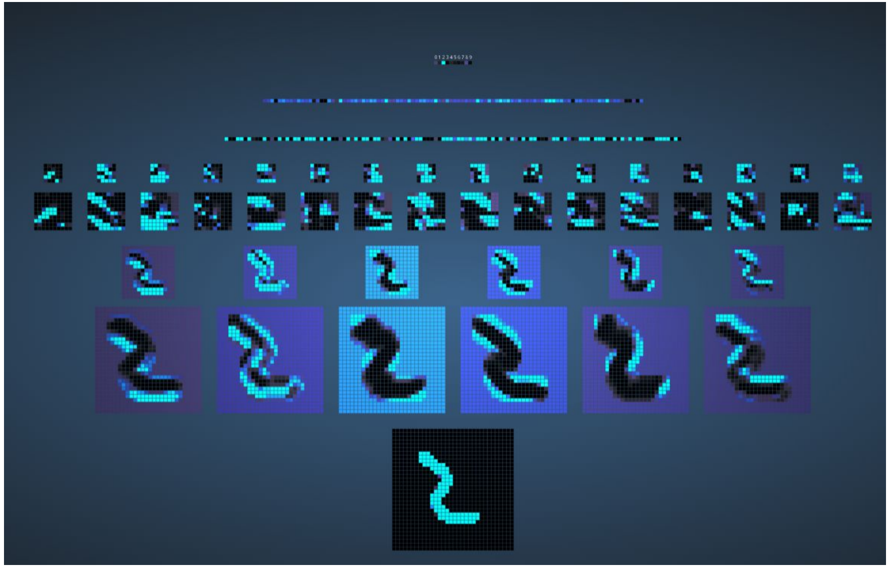
Convolutional Neural Network



- Going in deep with several conv-layers, the image size get reduced while the numbers of filters increase to distinguish more complex and high level features
- Then the resulting image is flattened to be fed into the last layers (Fully connected or Dense) that are traditional Multi-layer perceptron layers

Inside a CNN

- http://scs.ryerson.ca/~aharley/vis/conv/flat.html : Online simulator of number recognition with CNN, useful to understand what happens inside the network and which features get extracted by convolutions



CNN in Keras

Convolutional layer

```
from keras import models
from keras import layers

network = models.Sequential() #initialize the neural network

network.add(layers.Conv2D(N°of filters, (f, f), activation='relu', input_shape=(width, height, N°of channels))) # Convolution operator
network.add(layers.MaxPooling2D((fw, fw))) #Pooling operator
# For the other convolutional layers the input shape is automatically calculated but PAY ATTENTION
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))

network.add(layers.Flatten()) #Flatten the image
network.add(layers.Dense(N°neurons, activation='relu')) #Fully connected layers as in normal neural networks
network.add(layers.Dense(10, activation='softmax'))
network.summary() # Shows information about layers and parameters of the entire network
```

MNIST dataset

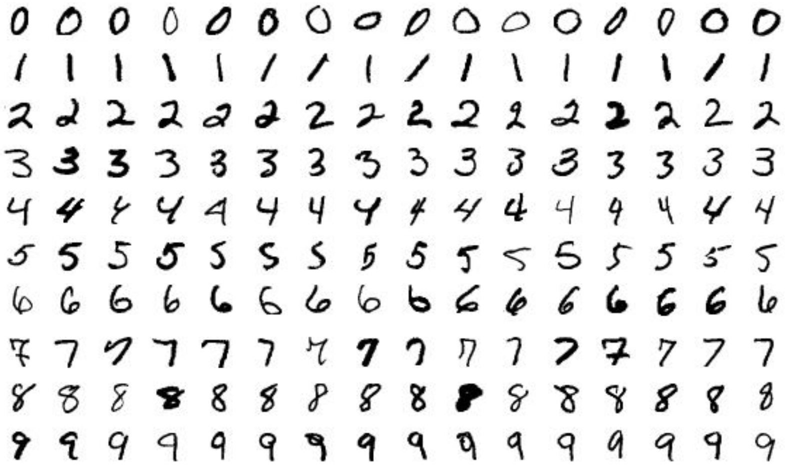
- Dataset of handwritten numbers

- 60k examples for training-set
- 10k examples for test-set

- Grey-scale 28x28 pixel images

- 10 classes for the classification task (0,1,2,3,4,5,6,7,8,9)

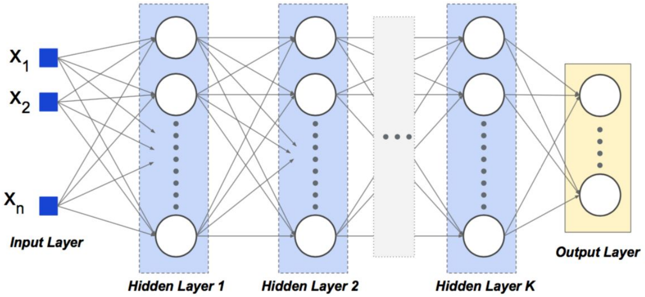
- Import in Keras with
  - from keras.datasets import mnist
  - (train_images, train_labels), (test_images, test_labels) = mnist.load_data())

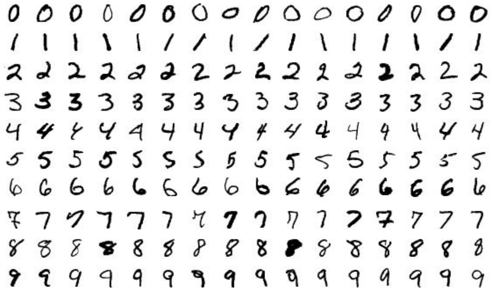


BREAK

Exercise 1

- Try to get the best accuracy as you can (at least 97% on test-set) on MNIST dataset with a multi-layer perceptron
  - The choice of the number of neurons and the number of hidden layers is free
  - It is suggested to start with an high number of neurons that decrease until the output layer
  - Try with 10 epochs and batch size = 128
  - ***NOTE: Images have to be flatten to be an input for the network, pay attention to the shape of the input vector (Go to see again slide 9)***
  - ***There is only 1 channel (Grey-scale image)***

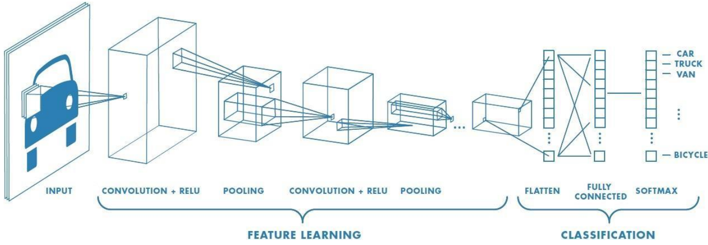


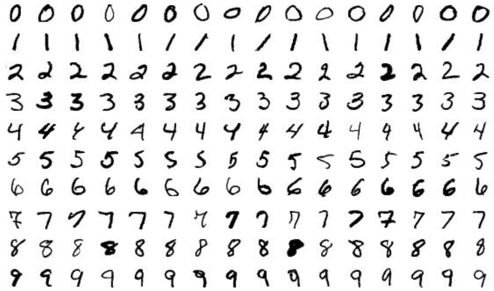


Exercise 2

- Repeat the previous exercise on MNIST dataset but with a Convolutional Neural Network trying to get a better result !
    - The choice of f (filter dimension), fw (pooling dimension) and the number of filters for each layer is free
    - It is suggested to start with a few numbers of filters and then increase it going in deep
    - Try with 5 epochs and batch size = 128

    - ***NOTE: Images have to be a tensor as an input for the network, pay attention to the shape of the tensor (Go to see again slide 29)***
    - ***There is only 1 channel (Grey-scale image)***





Benefits of Convolutional Neural Networks

- **Less parameters** respect to Multy-layer perceptron, risk of overfitting is reduced (but still a problem)

- **Parameters sharing**: Filters (as feature detector) convolve the entire image
  - The idea is that if a feature detector is useful in one part of the image, it is probably useful also in other parts of the image

- **Sparsity of connections:** In each layer, each output values depends only on a small number of inputs and not on the entire image! (Benefit due to the use of filters)

- Learning as an optimization problem the filters value, it is possible to learn feature detectors for example for vertical, horizontal and diagonal edges at the same time
  - The most important consequence is that **extracted features are Translational invariance**

How to choose parameters ?

- There is no a formal rule to select hyperparameters ! Design a Deep learning model is more like an art based on the experience.

- Some general tips:
    - Image shape shrinks going deeper in the network while the number of filters increase (but not the shape)

    - More conv-layers help the network to extract more robust feature. However if you do not have an huge dataset the overfitting risk is still high!

    - Do not underestimate the importance of **Pooling operator** to make feature more robust!

    - If you do not have a very huge dataset use the **Dropout** technique before the last layers

    - Last layers are usually dense (or fully connected) to help the network to classify

    - Hyperparameters can be estimated using also a **Grid search** or a **Random search**

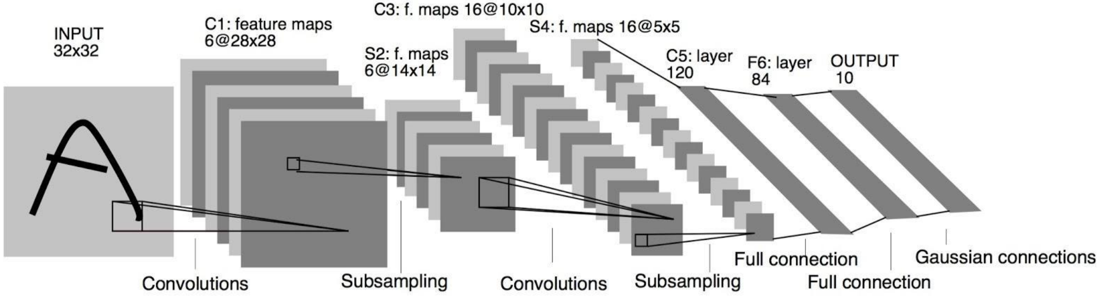
Existing CNN architectures in Deep learning literature

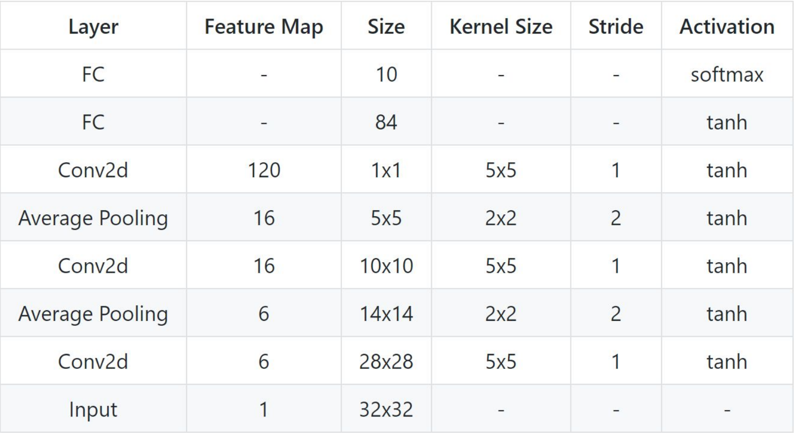
- LeNet-5 (1998 Yann LeCun)
- AlexNet (2012 Alex Krizhevsky, Geoffrey Hinton)
- GoogleNet (2014)
- VGGNet(2014 Simonyan and Zisserman)
- ResNet (2015 Kaiming He)

LeNet

- Used for the MNIST dataset by Yann LeCun
- 3 Convolutional layers (C1+S2 , C3+S4, C5)
- After C5 images get flatten in 120
- F6 is a dense layer as F7(output)

- Nowadays ReLU is more suggested than tanh as activation function and also MaxPooling instead of Average Pooling





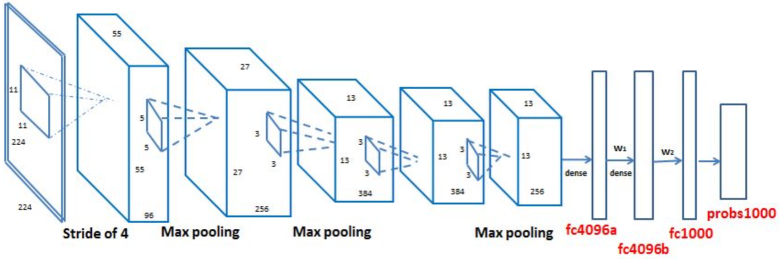
Layer	Feature Map	Size	Kernel Size	Stride	Activation
FC	-	10	-	-	softmax
FC	-	84	-	-	tanh
Conv2d	120	1x1	5x5	1	tanh
Average Pooling	16	5x5	2x2	2	tanh
Conv2d	16	10x10	5x5	1	tanh
Average Pooling	6	14x14	2x2	2	tanh
Conv2d	6	28x28	5x5	1	tanh
Input	1	32x32	-	-	-

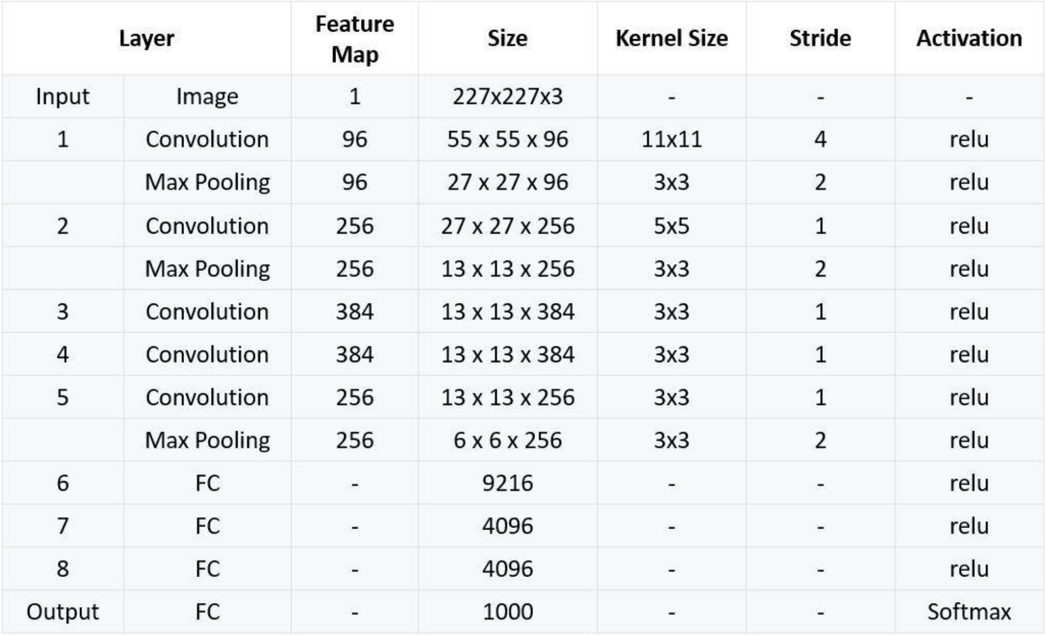
LeNet-5 in Keras

```
from keras import models
from keras import layers

network = models.Sequential() #initialize the neural network
network.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=input_shape, padding="same"))
network.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid'))
network.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
network.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
network.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
network.add(layers.Flatten())
network.add(layers.Dense(84, activation='tanh'))
network.add(layers.Dense(nb_classes, activation='softmax'))

network.summary() # Shows information about layers and parameters of the entire network
```

AlexNet

- More recent architecture





Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

VGGNet

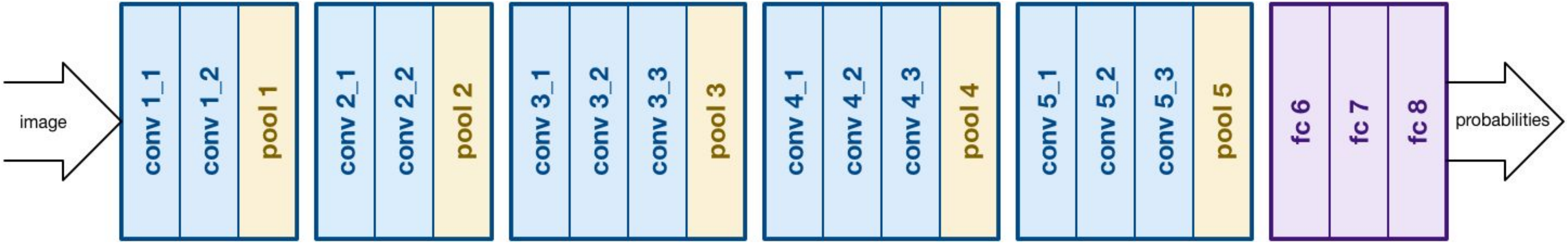
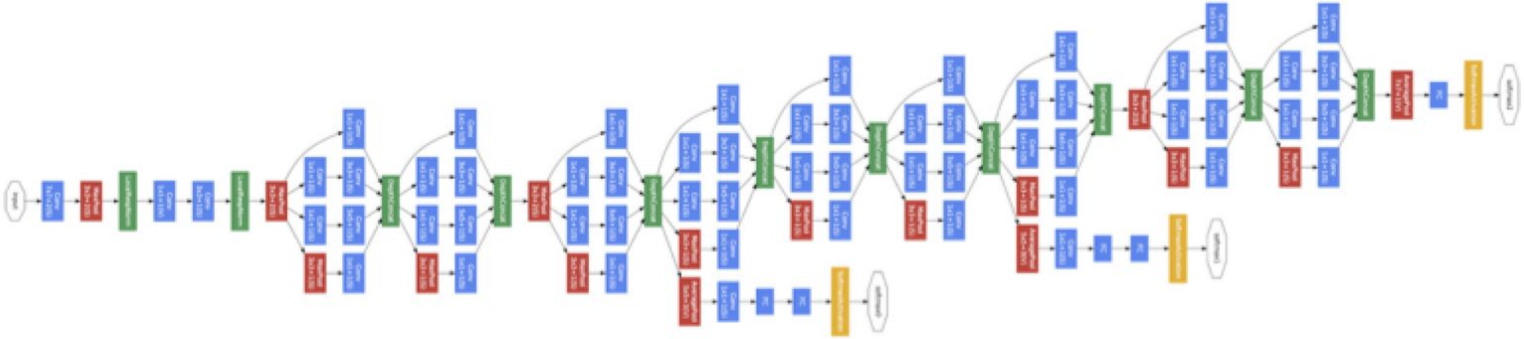


image
conv 1_1
conv 1_2
pool 1
conv 2_1
conv 2_2
pool 2
conv 3_1
conv 3_2
conv 3_3
pool 3
conv 4_1
conv 4_2
conv 4_3
pool 4
conv 5_1
conv 5_2
conv 5_3
pool 5
fc 6
fc 7
fc 8
probabilities

GoogleNet



Convolution

Pooling

Softmax

Other

Exercise: Pet classification

- Is it a cat (0) or a dog (1) ?

- Reduced dataset
  - 2k training-examples (1k for cats and 1k for dogs)
  - 1k test-set examples (500 hundred cats and the same for dogs)

- Data can be found in the dataset folder

- The function to import dataset is done only for the training

- **Goal:** Implement a CNN architecture trying inspiration from the existing architectures presented previously

- **Goal 2:** Try to have an accuracy on the validation set at least of 78/80%

Pet classification: tips and other requests

- Due to hardware constraints try first your architecture with 5 epochs and batch size = 128
  - When you are satisfacted try with 30 epochs or more (Pay attention it requires some minutes to train)

- With a small dataset there is an high risk of overfitting
  - Add a dropout level before fed the last fully connected layers
  - network.add(layers.Dropout(VALUE between 0 and 1))

- Be careful on input shapes and output shapes, at the same time the suggested filter dimension is 3x3
  - Do not use padding and stride options at the beginning
  - Images have 3 channels RGB, keep in mind in the choose of shapes

- Plot the history of training using test-set data with the validation_data options, it will help you in understanding how much the network is overfitting