CNN: Convolutional Neural Networks

Diane Lingrand and many contributors

UNIVERSITÉ
CÔTE D'AZUR

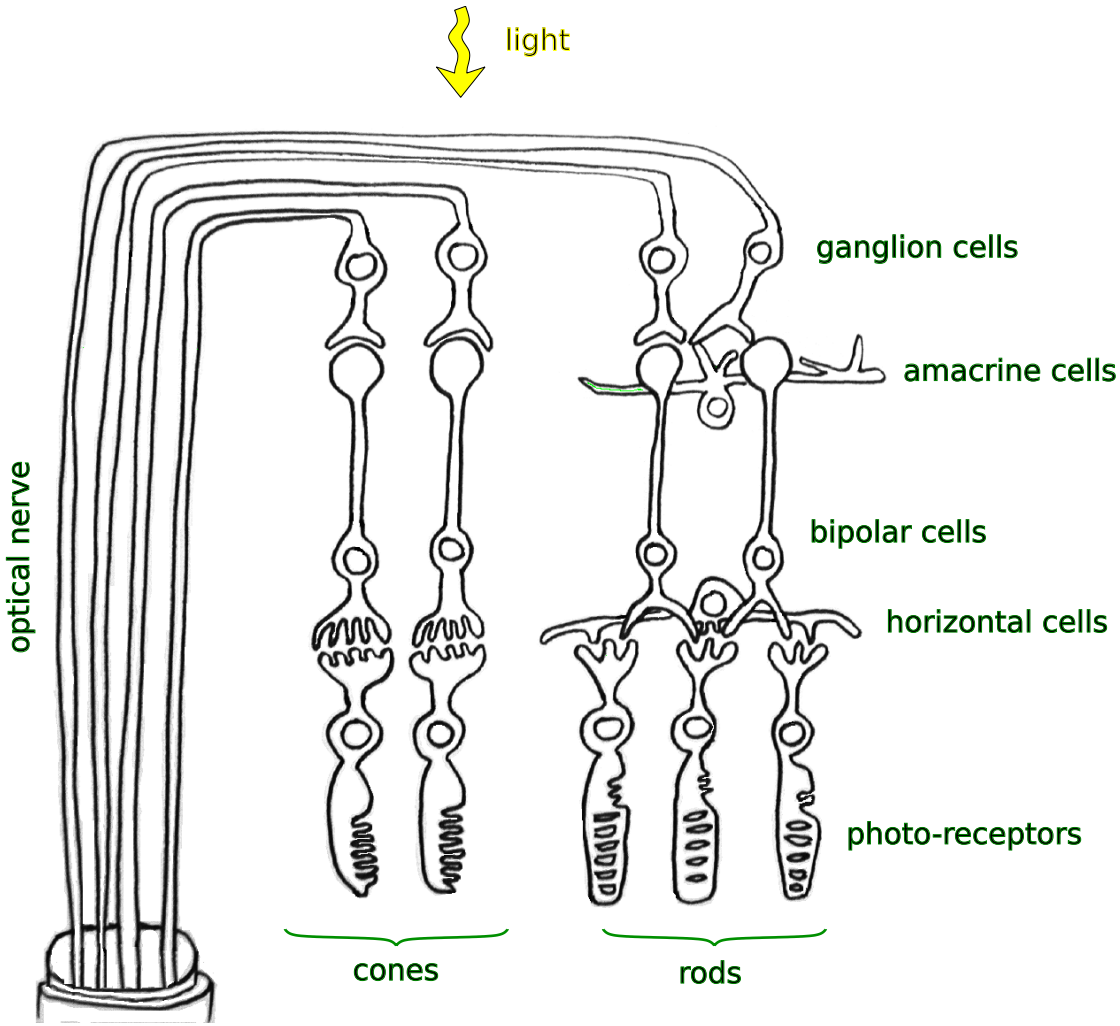
SI5

2022 - 2023

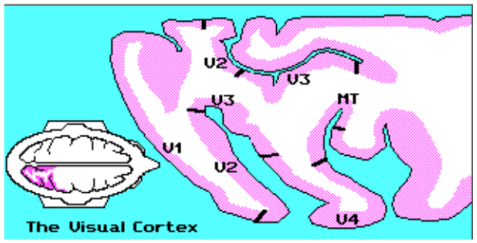
Outline

Deep Architecture in the Brain : the retina

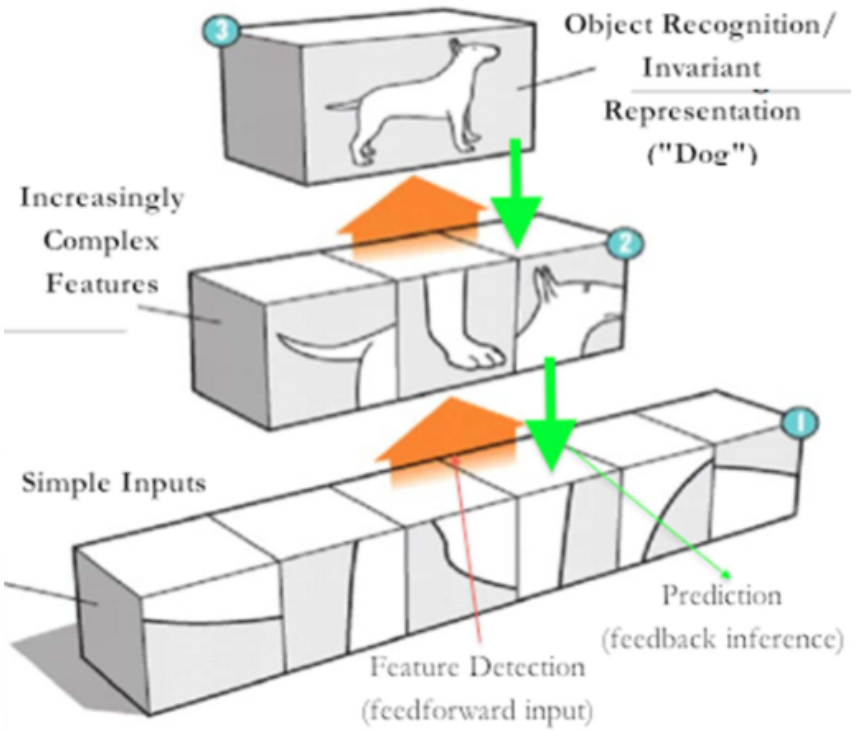
- photoreceptor cells : cones and rods
- bipolar cells (first neuron) : link to one or several photoreceptors
- ganglion cells (second neuron) link to one or several bipolar cells
- optical nerve (blind spot)



Deep Architecture in the Brain : the visual cortex



- area V1 : edge detectors (all directions)
- area V2 : primitive shape detectors
- area V3, V4 and more : higher level visual abstractions



Motivation

- image classification
  - neural network as a function for image classification
    - huge number of weights to learn (nb. layers x nb. neurons)
    - even with GPUs
  - inspired by old fashion image processing methods
    - 70's : Sobel, Laplace, Kirsch, Prewitt, Mean or Gaussian smoothing
    - 80's : power of two $\Rightarrow$ integer
    - 90's : SIFT, SURF, ...
    - all are based on convolution functions

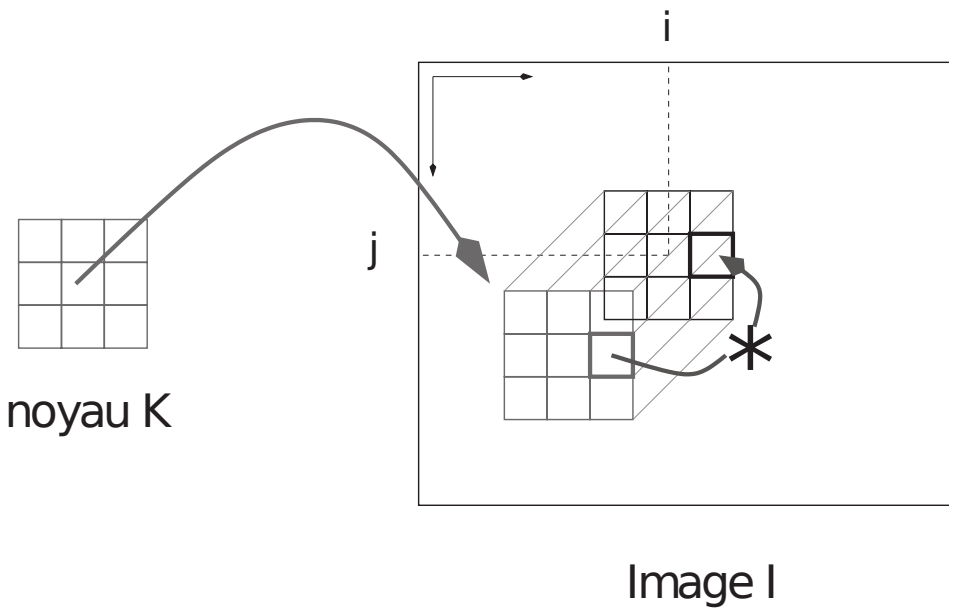
Back to convolution operator

- For f and g, discrete functions :

$$f * g(x, y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(x, y) g(u - x, v - y)$$

- Convolution of image I_1 by a kernel K of dimension $(2p + 1)x(2q + 1)$:

$$I_2[i][j] = \sum_{k=0}^{2p} \sum_{l=0}^{2q} I_1[i - k + p][j - l + q] K[k][l]$$

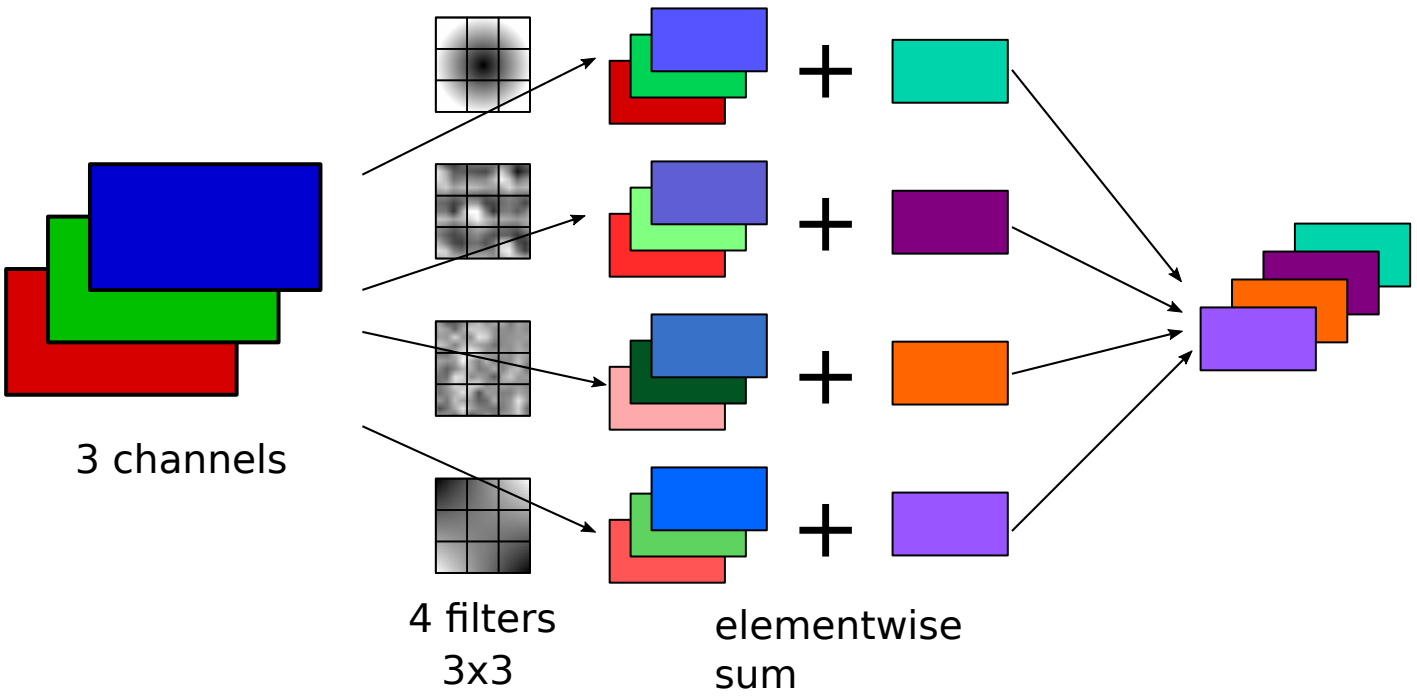


Idea of CNN : convolution layer

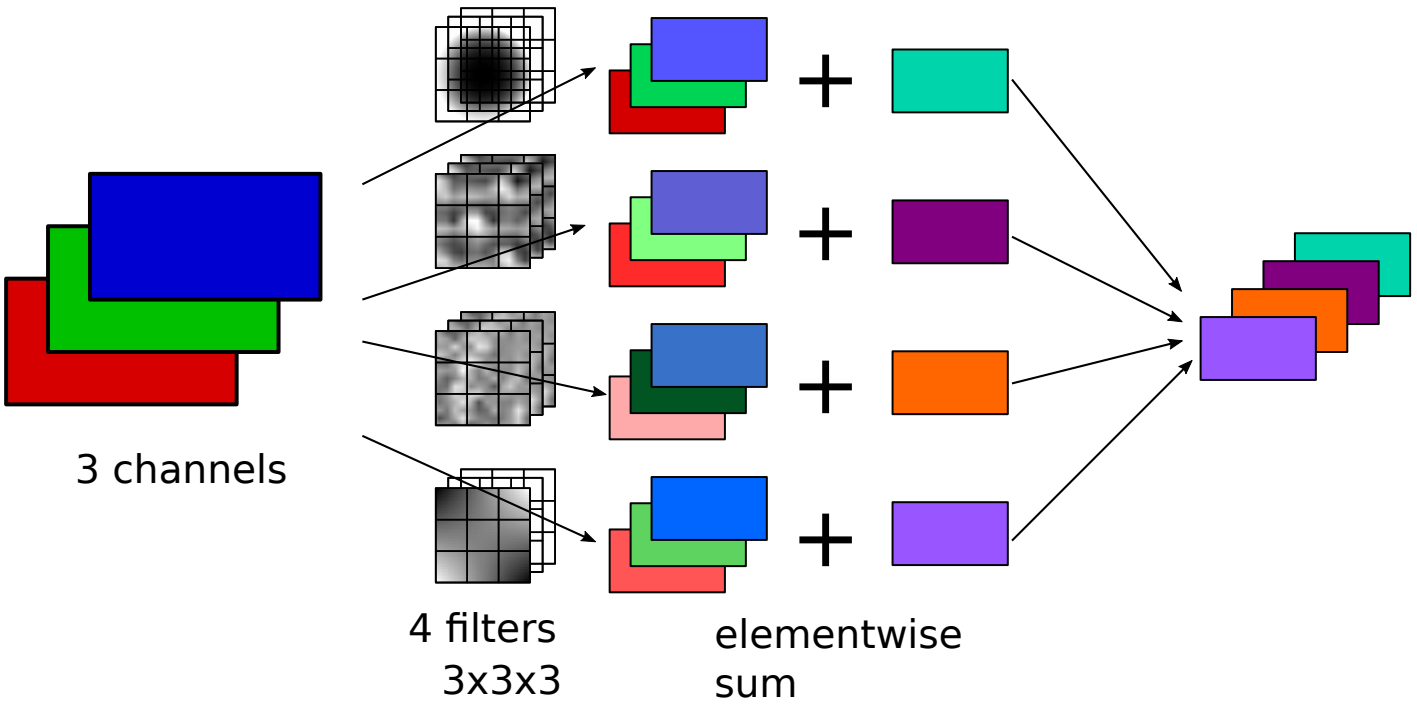
- learning the convolution filters
  - convolution : sum of weighted entries
  - 1 convolution filter = 1 neuron
  - the weights are the filter coefficients
- sharing weights with different neurons
  - the same convolution is applied to every pixels in the image
  - less weights to learn !
  - example : one 3x3 filter : 10 coefficients (9 for filter + 1 for bias)
- in keras : `Conv2D` instead of `Dense`
  - parameters : number of filters, size of filters (usually $k * k$), stride
  - input : size of image and number of channels (3 if RGB)
  - output : tensor of dimension nb. filters, (new) dim of images
  - `Conv1D` and `Conv3D` also exists

More details on dimensions for a `Conv2D` layer

- parameters :
  - input : tensor of dimension $w \times h \times ch$
  - filters : nb filters, dimension $k \times k$
  - output : tensor of dimension $w' \times h' \times nb$
- question : how did the number of channels ch dimension vanished ?
  - for each filter, elementwise addition of the result of the convolution of each channel by this filter



More details on dimensions for a `Conv2D` layer



- number of parameters to learn :
  - each filter : 3x3x(number of input channels) +1
  - 4 filters
  - thus 4x(3x3x3 + 1) = 112

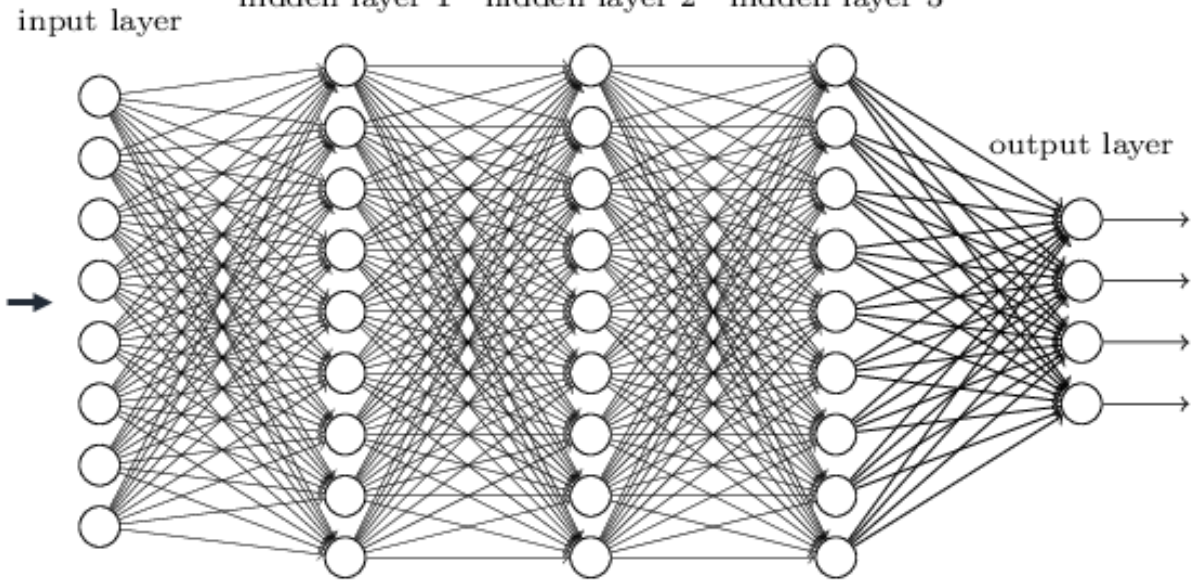
See `https://arxiv.org/pdf/1603.07285v1.pdf` for a complete discussion about dimensions

CNN : Convolutional Neural Network

- many convolutional layers
  - in order to extract the characteritics of images
  - but not only convolutional layers :
    - pooling
    - flatten
  - ReLU activation or Leaky ReLU ...
- two hidden fully connected layers at the output
  - the function of classification

CNN : Representative layers

Deep neural networks learn hierarchical feature representations



Activation functions

- Main activation functions :
  - Sigmoid
  - ReLU : Rectified Linear Unit
  - Leaky ReLU : in order to avoid exploding gradients
  - Softmax : transforms output values into values that can be interpreted as probabilities
- Other activation functions :
  - tanh, LeakyReLU, PReLU, ELU, SELU, GELU ...

Softmax layer

- smooth approximation to the arg max function
  - assign probabilities to indices of having the highest response $z = [z_0 z_1 z_2 ... z_n]$ for each i,

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  - every value in [0 1] and the sum is equal to 1

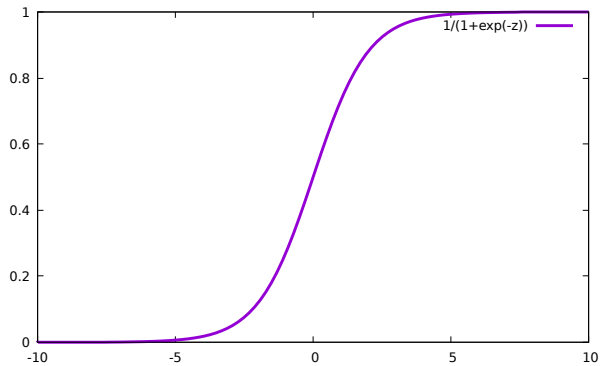
Why ReLU ?

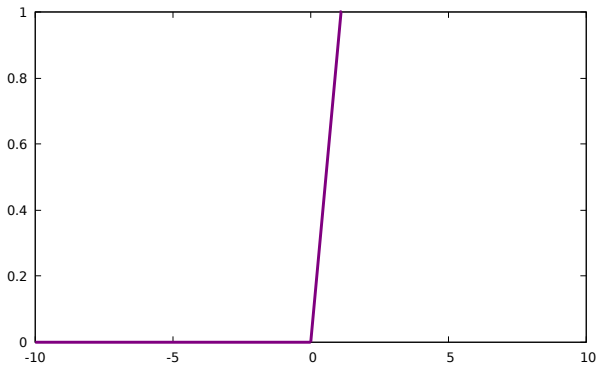
- problem of vanishing gradient
  - remember the delta rule and gradient descent :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^{n} w_{ik}\delta_{ik}$$

and

$$\forall j \in [0\ m]\ w_j \leftarrow w_j - \eta\delta_i y_{ij}$$





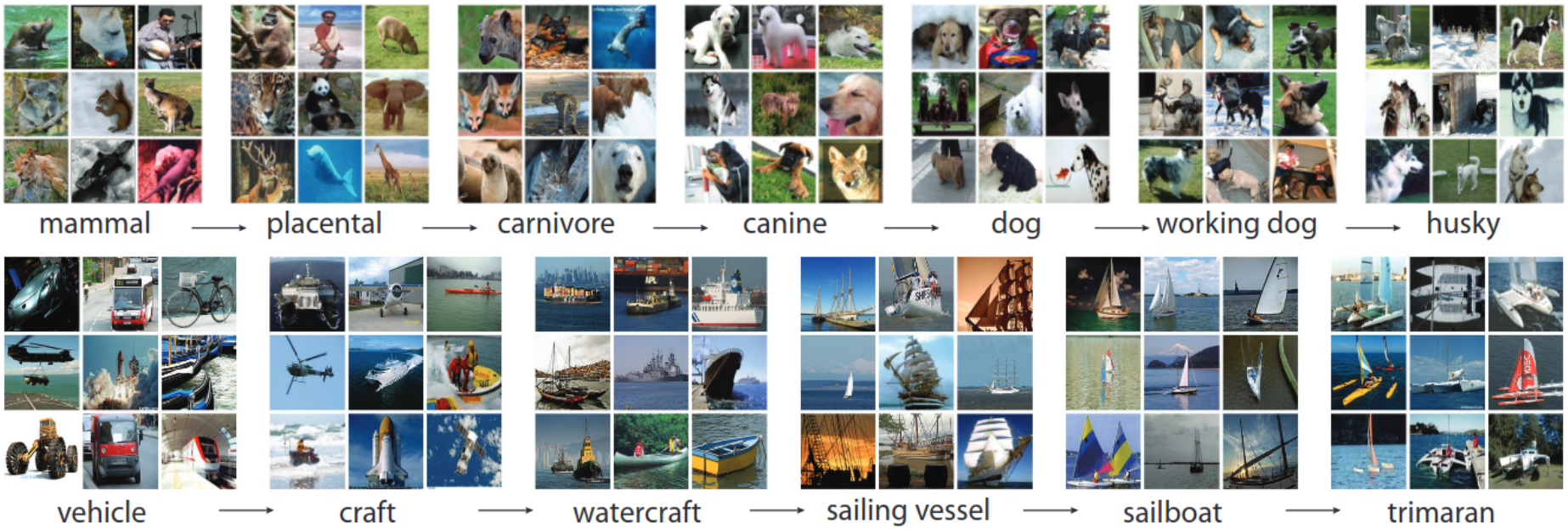
- different solutions :
  - change to activation function to ReLU
  - residual networks
  - batch normalisation

Pooling layers

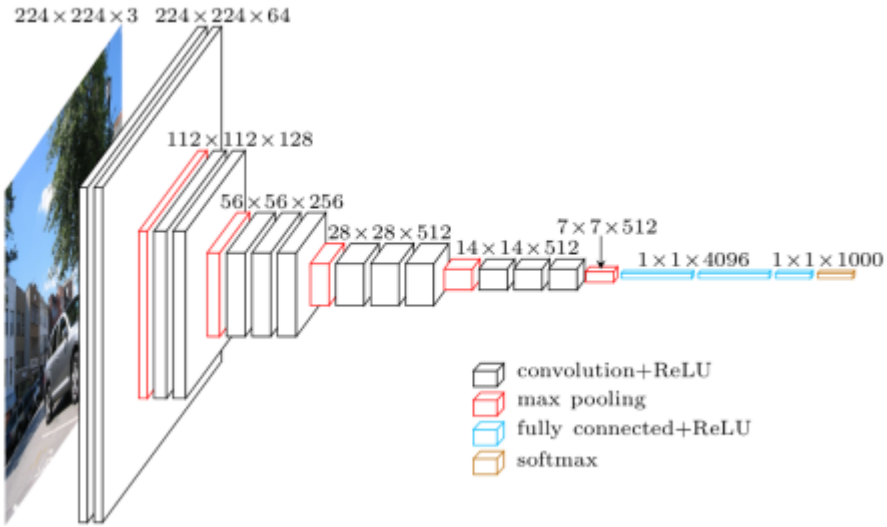
- usually after a convolutional layer
- reduces the size of data (downsampling)
- type of pooling
  - max pooling `from keras.layers import MaxPooling2D`
  - average pooling `from keras.layers import AveragePooling2D`
- dimension of pooling
  - default : 2X2 with stride 2
  - dimension
  - stride : factor of downsampling
  - padding ('VALID' means discard borders, 'SAME' zero padding)
- also exists : global pooling
  - max and average
  - reduces the dimensions to (batchSize, channels)

ImageNet

- more than 1000 classes
- 50 million labeled images
- in the ILSRVC : ImageNet Large Scale Visual Recognition Challenge



VVG-16, VGG-19, 2014



- deep
  - VGG-16 : 16 refers to the number of weighted layers (conv or dense)
  - VGG-19 : 19 layers to learn
- small convolution filters (less parameters)
- Simonyan and Zisserman (University of Oxford and Google Deep Mind) :`https://arxiv.org/pdf/1409.1556.pdf` - ImageNet Challenge (ILSVRC) 2014

Topology of VGG16

- (9*3+1)*64 = 1792
- (9*64+1)*64 = 36928
- ...

```
VGGmodel = VGG16(weights='imagenet', include_top=False)
VGGmodel.summary()
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, None, None, 3)]   0
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
```

Applications

- image classification
- image descriptor
  - output of convolution layer as a vector describing an image
- transfert learning
  - learning only the last fully connected layers for a new image classification task

Image descriptor using keras

Exemple from ExtractfeatureswithVGG16 :

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Spectrogram