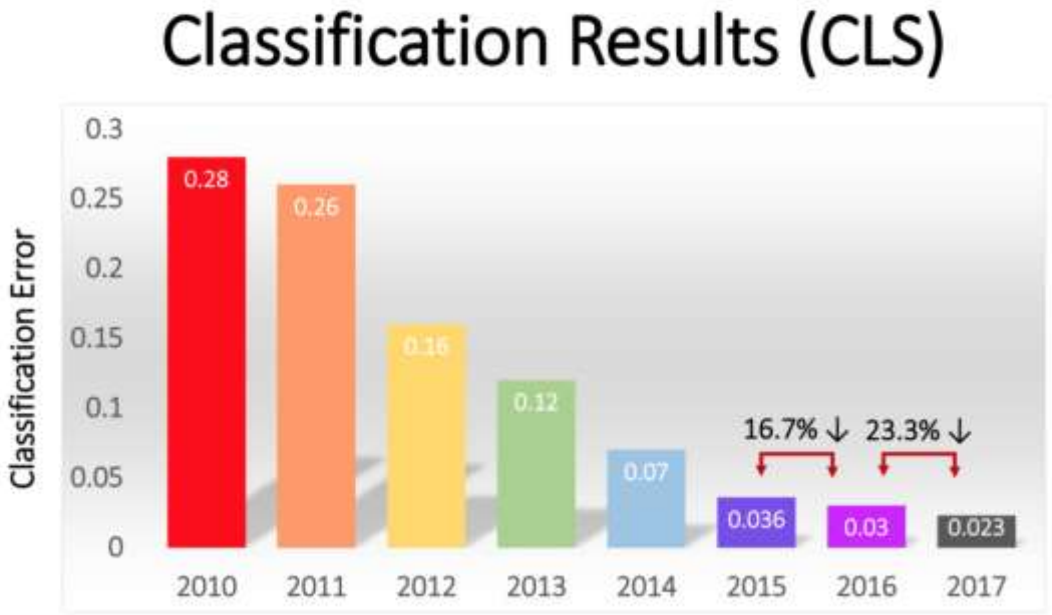
# Computer Vision

**Modelisation**

# Plan

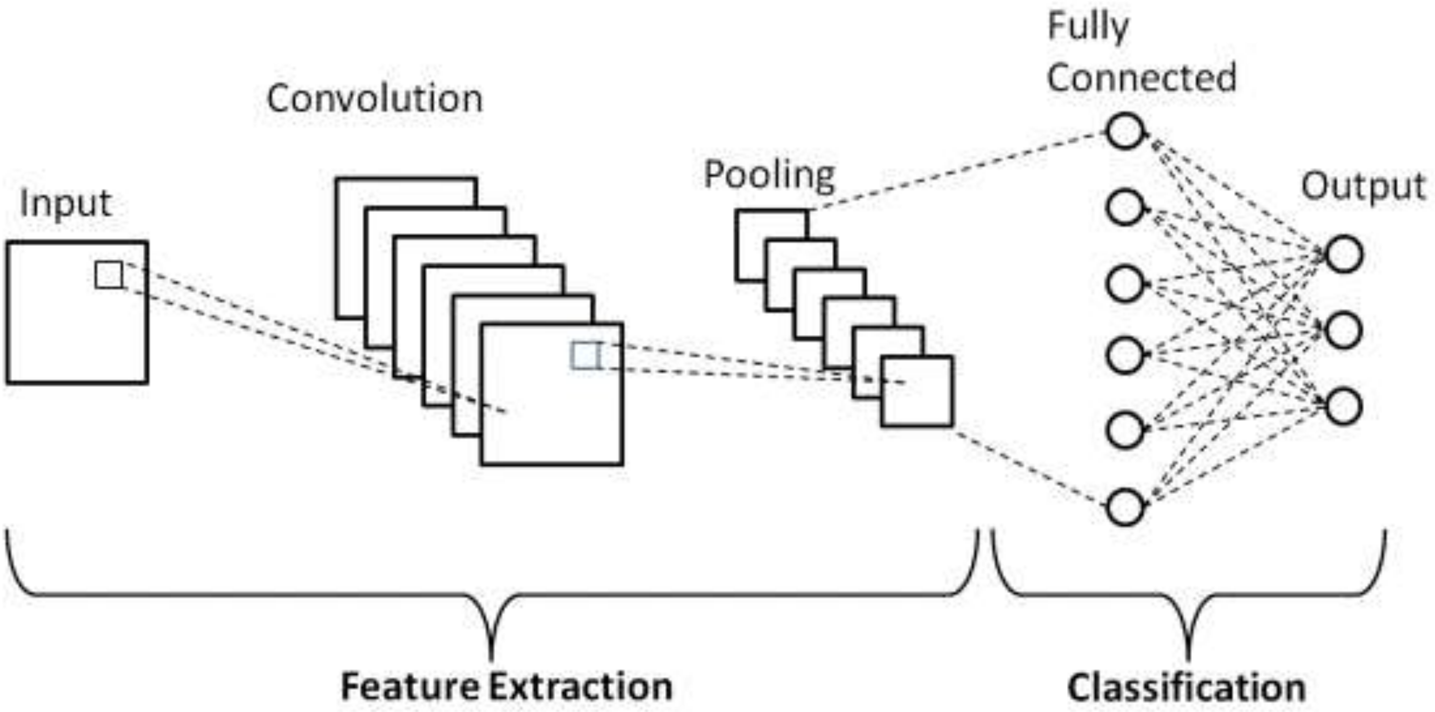



- **CNN**
  - Convolution layer
  - Pooling layer
  - Fully-connected layer
- **Transfert learning**
  - Purpose
  - Strategies
- Performance evaluation
  - Hyperparameters
  - Fitting

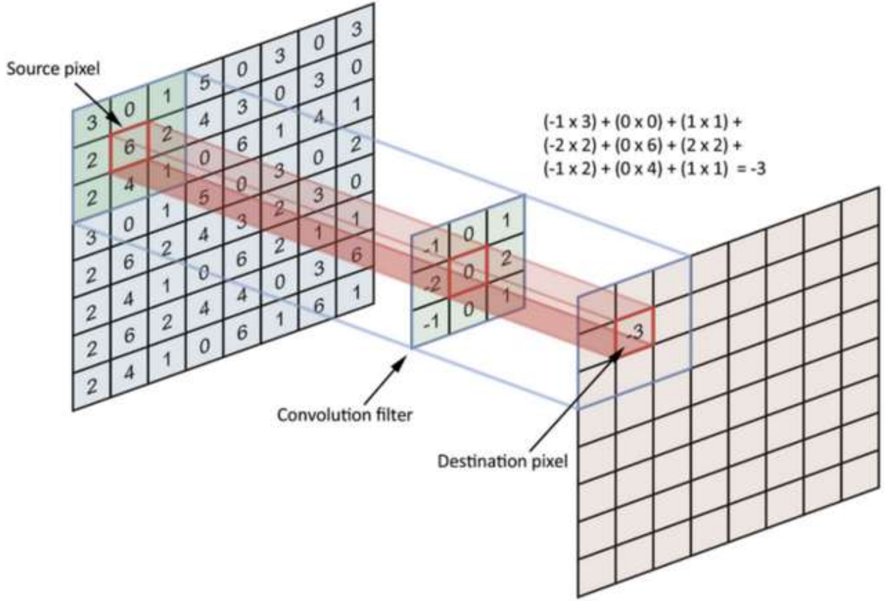
# CNN

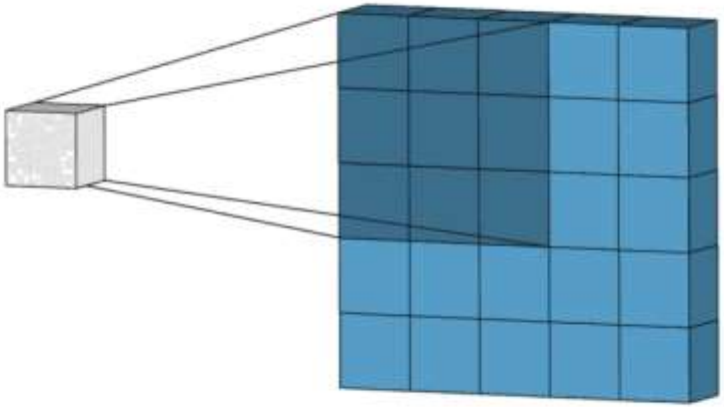
“Learn” the features

# CNN


Input
Convolution
Pooling
Fully Connected
Output
Feature Extraction
Classification

# CNN: convolution layer



**Activation function:**

- ➢ ReLU: f(x) = Max(0, x)

# CNN: convolution layer

**Image (input):**

- W : width (pixels)
- H : hight (pixels)
- D : number of channels

**Feature maps (output):**

- Wc : (W - F + 2P) / S + 1
- Hc : (H - F + 2P) / S + 1
- Dc : K

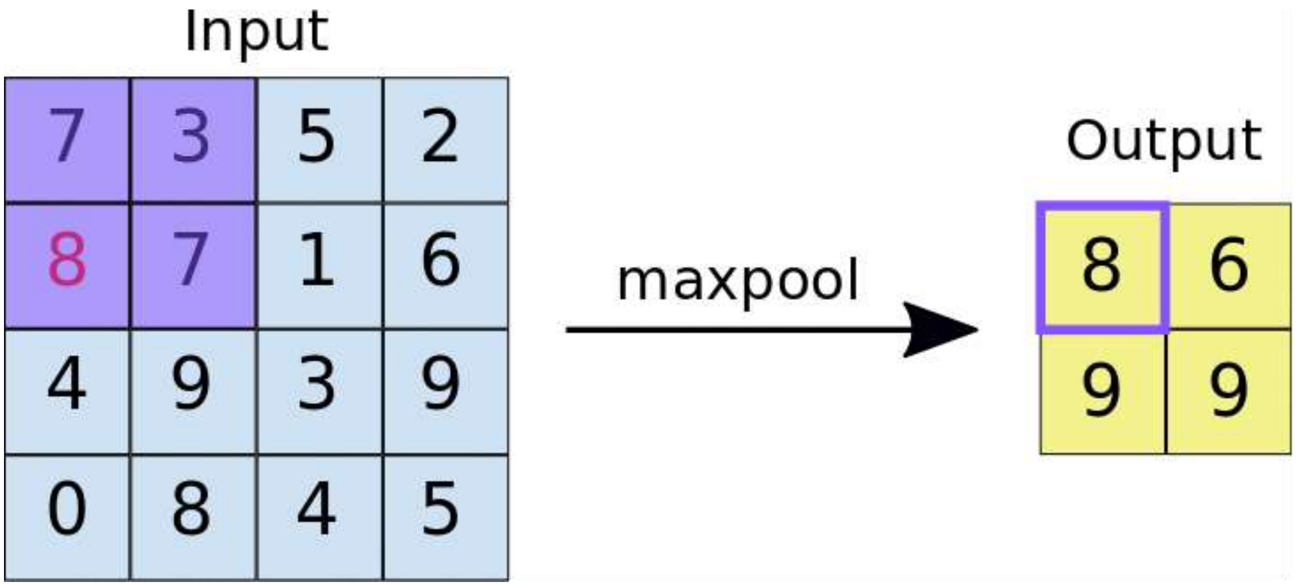
**Hyperparameters:**

- K : number of filters
- F : filter size in pixels (F x F x D)
- S : sliding step (pixels)
- P : 0 padding (pixels)

**Common choice:**

- F = 3 or 5
- S = 1 or 2
- P = 1

# CNN: pooling layer




**Advantages:**

- ➢ Dimension reduction
- ➢ Less sensitive to feature position

# CNN: pooling layer

**Feature maps (input):**

- W : width (pixels)
- H : hight (pixels)
- D : previously used number of filters

**Feature maps (output):**

- Wp : (W - F) / S + 1
- Hp : (H - F) / S + 1
- Dp : D

**Hyperparameters:**

- F : cell size (F x F pixels)
- S : pixels between cells

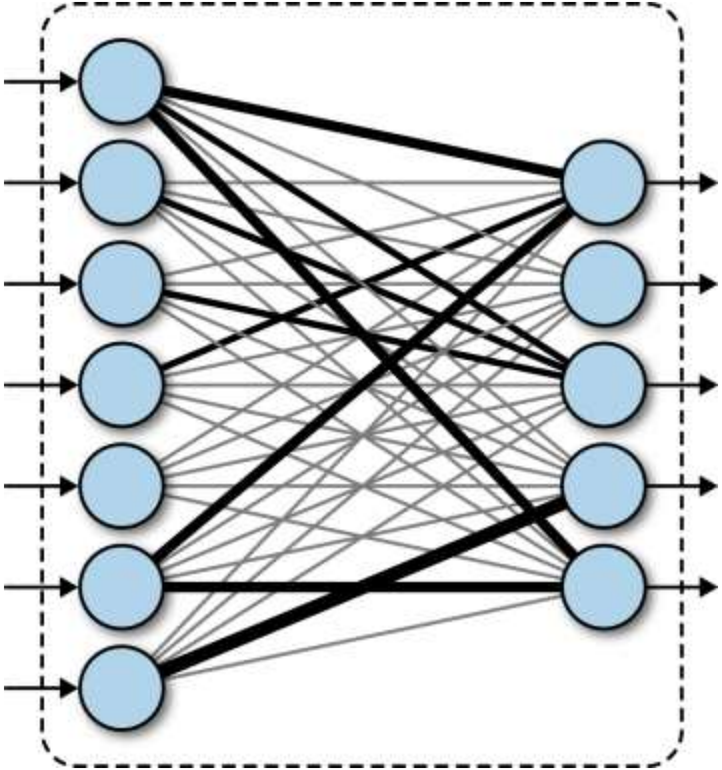
**Common choce:**

- F = 2 or 3
- S = 2

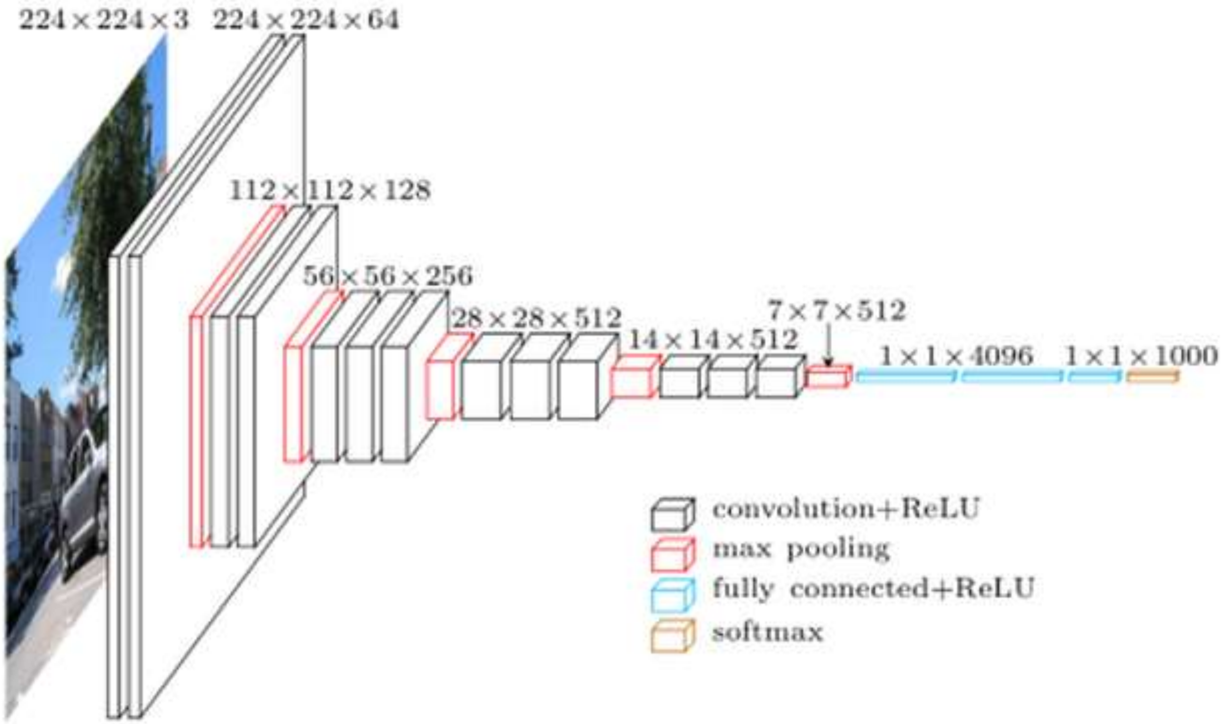
# CNN: fully-connected layer

**Options:**

- ➢ Flatten: convert output to 1D vector
- ➢ Number of neurons
- ➢ Dropout: rate
- ➢ Activation:
  - ○ ReLU
  - ○ Classifier 2N: logistic regression
  - ○ Classifier xN: Softmax



# CNN: VGG16



224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

# CNN: VGG16

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Empty NN
my_VGG16 = Sequential()

##########################
### Convolution block ###
##########################

# Add the first CNN layer with relu activation
my_VGG16.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), padding='same', activation='relu'))

# Add the second CNN layer with relu activation
my_VGG16.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

# Add the first pooling layer
my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# Repeat the previous steps as often as needed...

##############################
### Fully-connected block ###
##############################

# Convert 3D matrix to 1D vector
my_VGG16.add(Flatten())

# Add first fully-connected layer with relu activation
my_VGG16.add(Dense(4096, activation='relu'))

# Add second fully-connected layer with relu activation
my_VGG16.add(Dense(4096, activation='relu'))

# Add last fully-connected layer which is the classifier
my_VGG16.add(Dense(1000, activation='softmax'))
```

# Transfer learning

Time to be lazy and... smart ! :)

# Transfer learning: purpose

**Weights:**

- ➢ Convolution layer:
  - F : filter size
  - K : number of filters
  - weights = F x F x K + K
- ➢ Fully-connected:
  - W : width of input
  - H : height of input
  - K : number of filters
  - N : number of neurons
  - Weights = W x H x K x N + N

**VGG16 = 138 357 544 weights !!!**

# Transfer learning: strategies

**The idea:**

- **Keep the architecture**
- **Replace the classifier**

**Strategies:**

- Fine-tuning total:
  - Train all weights
  - Large dataset
- Feature extraction:
  - Fix all weights
  - Beside the ones of the classifier
  - Small / similar dataset
- Fine-tuning partial:
  - Fix some of the weights
  - Most likely in the first layers
  - Small / different dataset

# Transfer learning: exemple

```python

###########################
### Pre-trained exemple ###
###########################

from keras.applications.vgg16 import VGG16

# Get VGG16 available in keras
model = VGG16()

from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input

# Load image and reshape image to match VGG16 dimensions expectation
img = load_img('cat.jpg', target_size=(224, 224))

# Convert to numpy array because keras processes images as such
img = img_to_array(img)

# A CNN expects (in general) a collection of images.
# Reshape the array to add the number of images
img = img.reshape((1, img.shape[0], img.shape[1], img.shape[2]))

# Preprocess images the same way the images used to train VGG16 were preprocessed.
img = preprocess_input(img)

# Predict the class (or label) of this image
y = model.predict(img)

```

# Transfer learning: exemple

```python
##########################
### Transfer learning ###
##########################

from keras import Model

# Load the VGG16 trained on imagenet images without the fully-connected layer
model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

# Get the output of VGG16
x = model.output

# Add your new classifier
predictions = Dense(10, activation='softmax')(x)

# Design your new model
new_model = Model(inputs=model.input, outputs=predictions)

# Strat n°1:
for layer in new_model.layers:
    layer.trainable = True

# Strat n°2:
for layer in new_model.layers:
    layer.trainable = False

# Strat n°3: let's not train the first 5 layers (the largests)
for layer in new_model.layers[:5]:
    layer.trainable = False

# Compile the new model
new_model.compile(loss="categorical_crossentropy", optimizer=optimizers.SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])

# Train it
model_info = new_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=2)
```

# Performance evaluation

The suspense is intense !

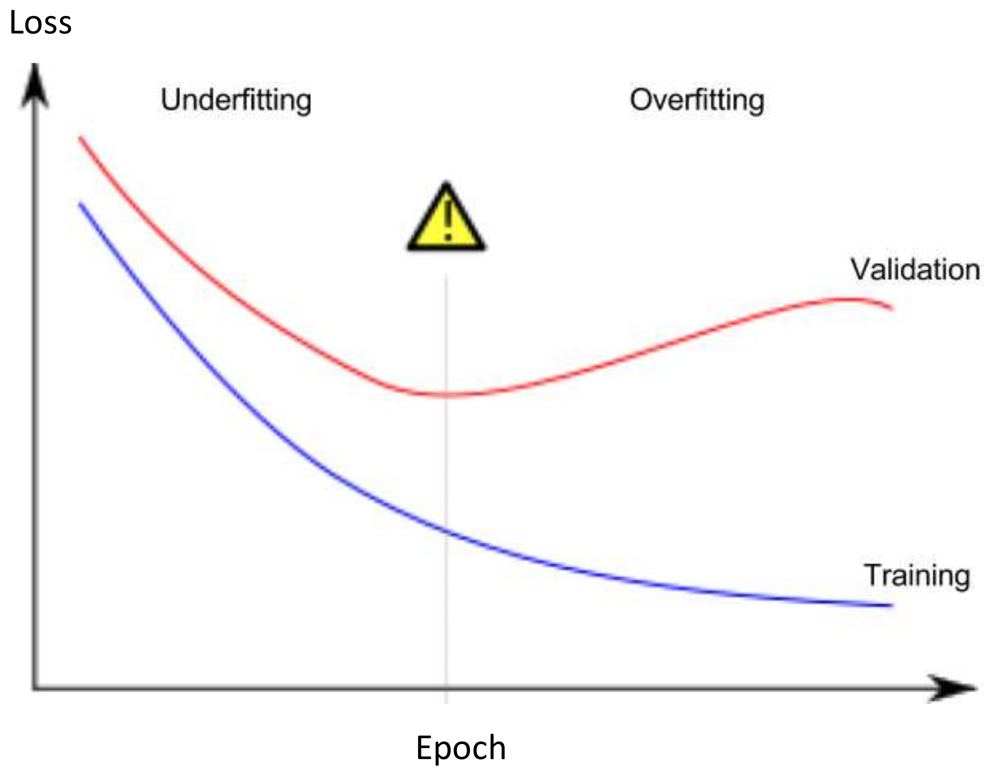
# Hyperparameters

- Loss function
- Optimizer:
  - Learning rate
  - Momentum
- Metrics

```python
##########################
### Transfer learning ###
##########################

from keras import Model

# Load the VGG16 trained on imagenet images without the fully-connected layer
model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

# Get the output of VGG16
x = model.output

# Add your new classifier
predictions = Dense(10, activation='softmax')(x)

# Design your new model
new_model = Model(inputs=model.input, outputs=predictions)

# Strat n°1:
for layer in new_model.layers:
    layer.trainable = True

# Strat n°2:
for layer in new_model.layers:
    layer.trainable = False

# Strat n°3: let's not train the first 5 layers (the largests)
for layer in new_model.layers[:5]:
    layer.trainable = False

# Compile the new model
new_model.compile(loss="categorical_crossentropy", optimizer=optimizers.SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])

# Train it
model_info = new_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=2)
```

# Underfitting vs. Overfitting

```
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```



# Practice

You know the drill

# Ai design: dataset / concept

- ➢ Select a dataset or an app idea
- ➢ Create your team !
- ➢ Define your strategy
- ➢ Prototype and test


Google Dataset Search Beta

kaggle

OpenML