Computer Vision

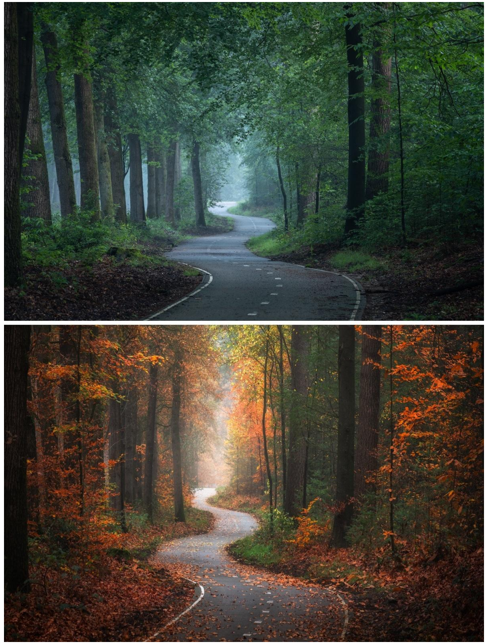
Modelisation

Reminder: purpose

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and **deep learning** models, machines can accurately **identify and classify objects** — and then **react** to what they "see."

- ➢ Object detection
- ➢ Object tracking
- ➢ Image segmentation / search
- ➢ 3D scene modeling

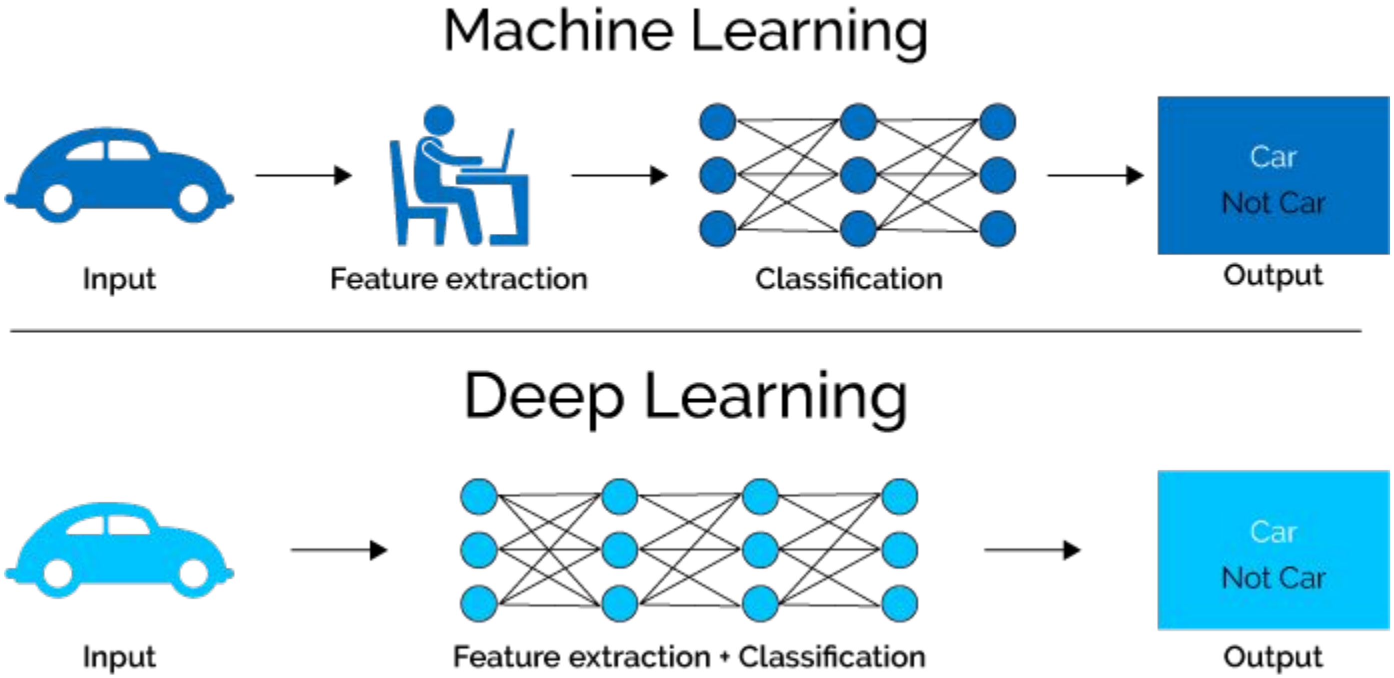
Reminder: problems



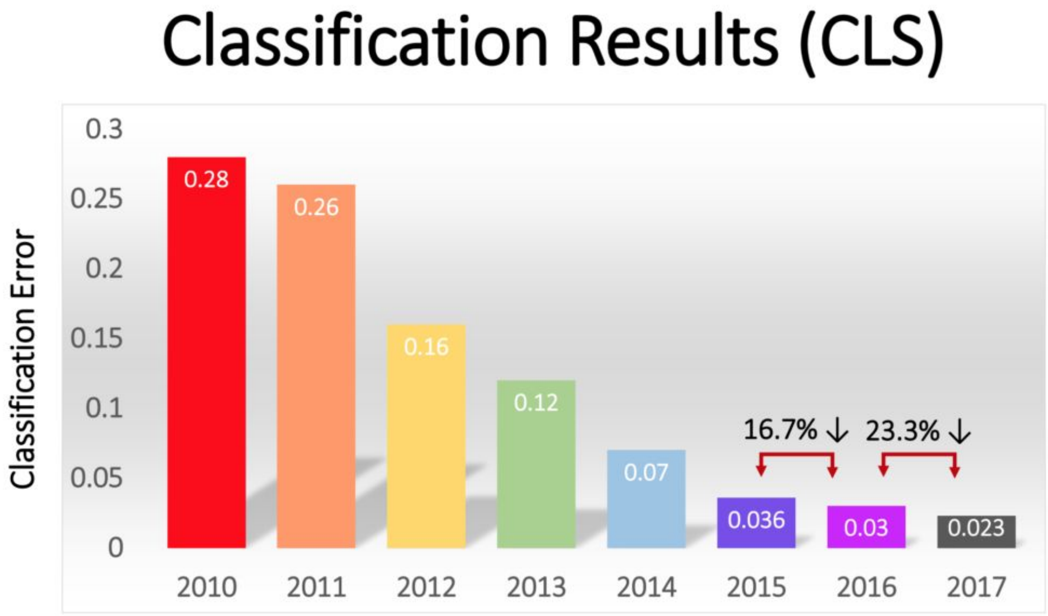
- ➢ Object orientation
- ➢ Lighting conditions
- ➢ Obstruction
- ➢ …

➔ **Too many possibilities !**

Why deep learning ?



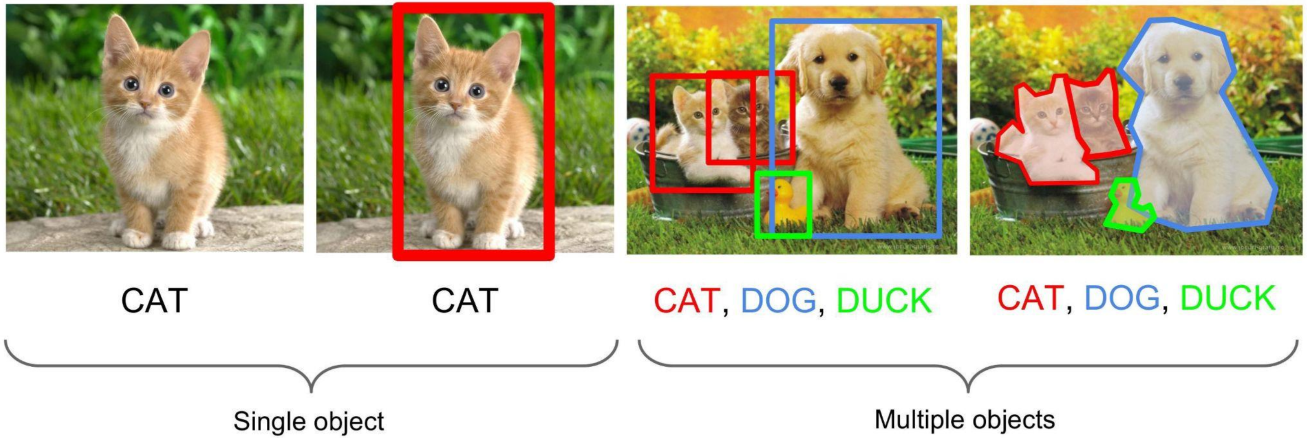
Plan



- ➢ Tasks
- ➢ CNN
- ➢ Compilation
- ➢ Transfert learning

Tasks

Classification - detection - segmentation - creation



Classification

CAT

Particularity:

- N possible outputs with N = number of classes.

Algorithms

- Mark 1
- SVM
- CNN

Datasets

- ImageNet
- CIFAR 100
- ISIC
- MURA
- DermNet

Models

- LeNet5
- VGG
- GoogleNet / MobileNet
- RestNet

Object detection

CAT

Particularity:

- Is an object present ?
- Location:
    - x and y (top left of the object area, or center)
    - Area height and width

Algorithms

- Sliding window
- CNN

Datasets

- ImageNet
- COCO
- PASCAL

Models

- R-CNN
- YOLO
- SSD

Image segmentation

CAT, DOG, DUCK

Particularity:

- Is an object present ?
- Object boundaries
- Give a value for each pixel

Algorithms

- CNN

Datasets

- COCO
- PASCAL
- BraTS
- Agriculture-Vision
- SpaceNet

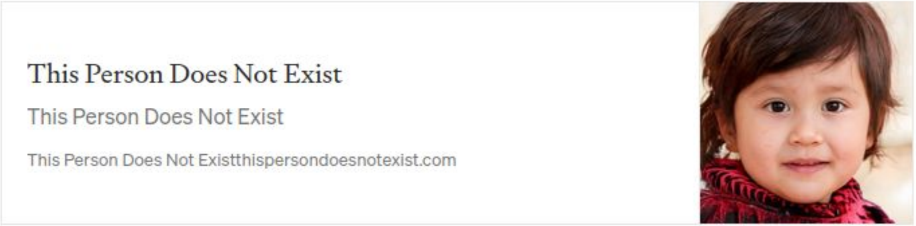
Models

- U-net
- Mask R-CNN

Image creation / generation

Particularity:

- Give a value for each pixel
- Unlike any training image
- Improve resolution



Algorithms

- CNN

Datasets

- Any...

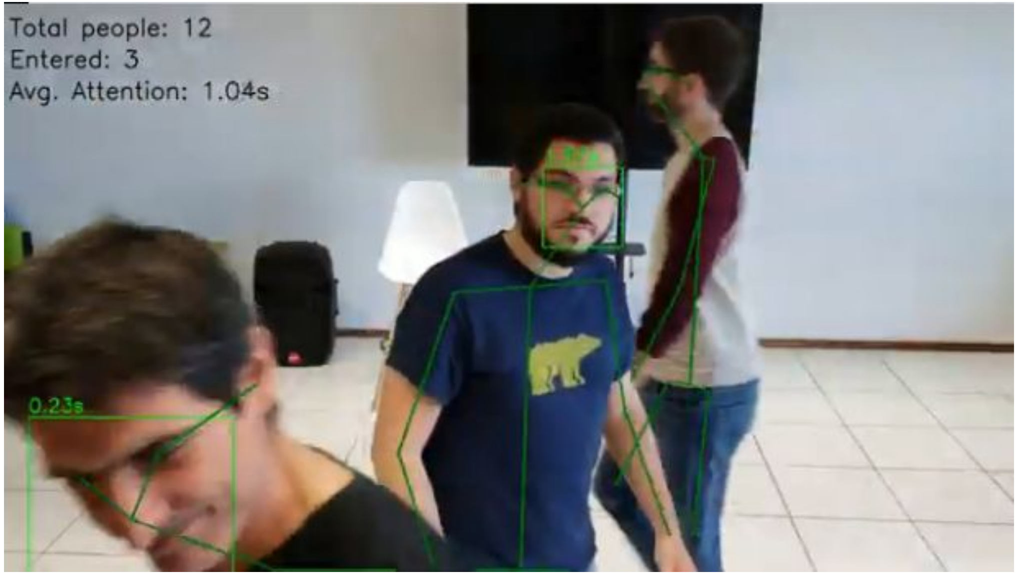
Models

- GANs
- VAEs

Potential

Behavioral tracking

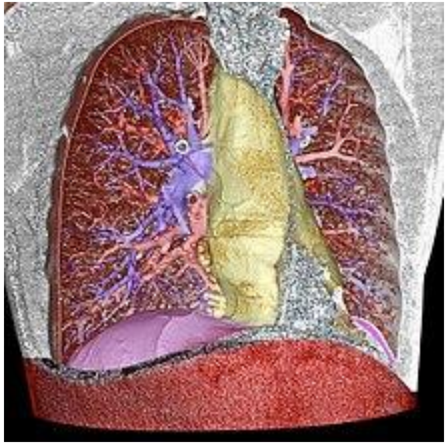
- ➢ Biomédical
- ➢ Robotics
- ➢ Airports
- ➢ Security / Surveillance
- ➢ Space exploration
- ➢ Customer experience
- ➢ ...



Potential

Healthcare

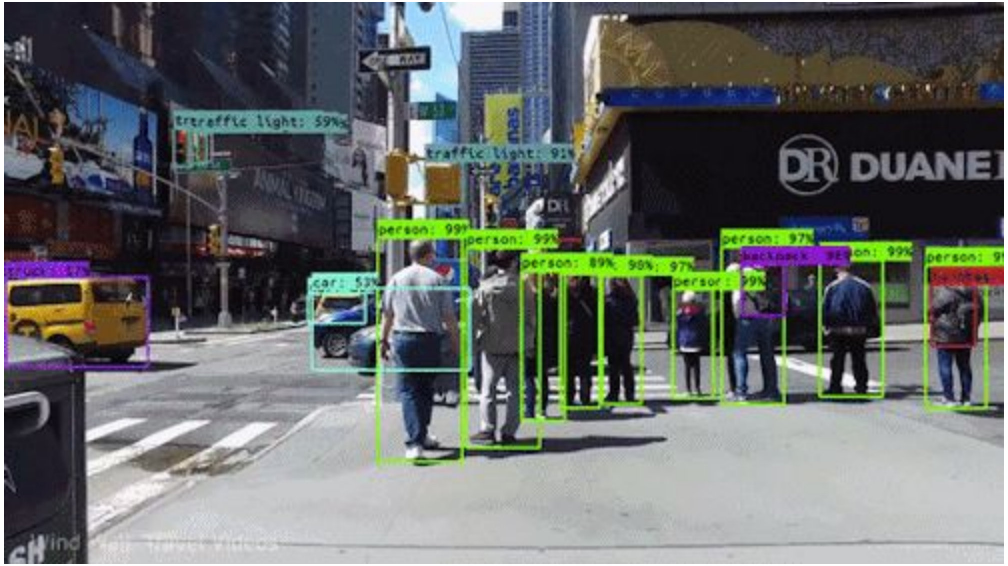
- ➢ Biomédical
- ➢ Robotics
- ➢ Airports
- ➢ Security / Surveillance
- ➢ Space exploration
- ➢ Customer experience
- ➢ ...



Potential

- Biomédical
- Robotics
- Airports
- Security / Surveillance
- Space exploration
- Customer experience
- ...

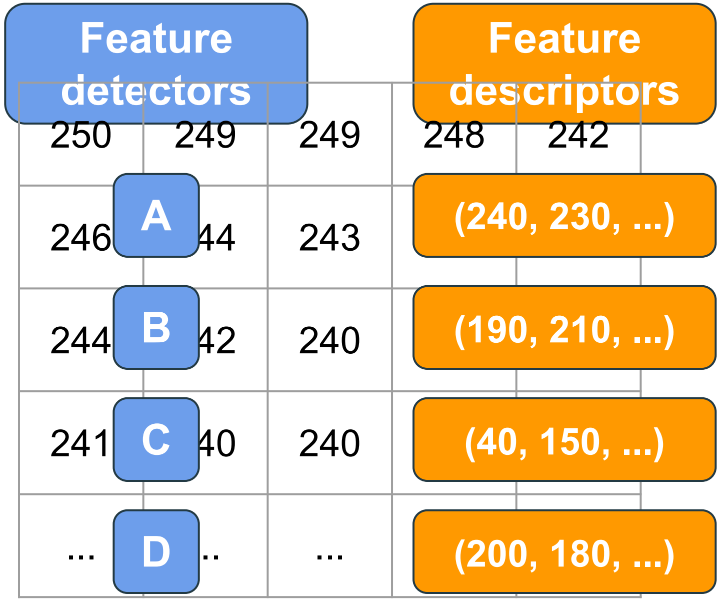
Autonomous vehicles

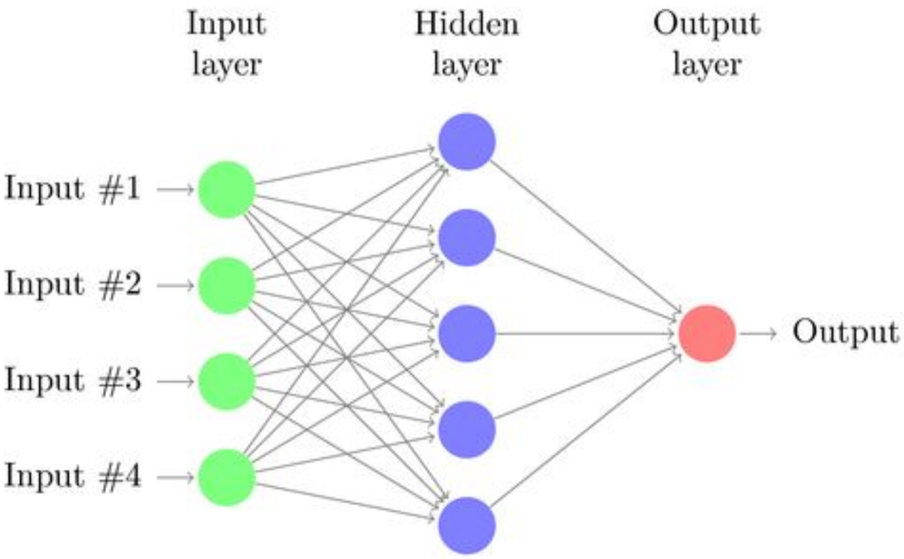


CNN

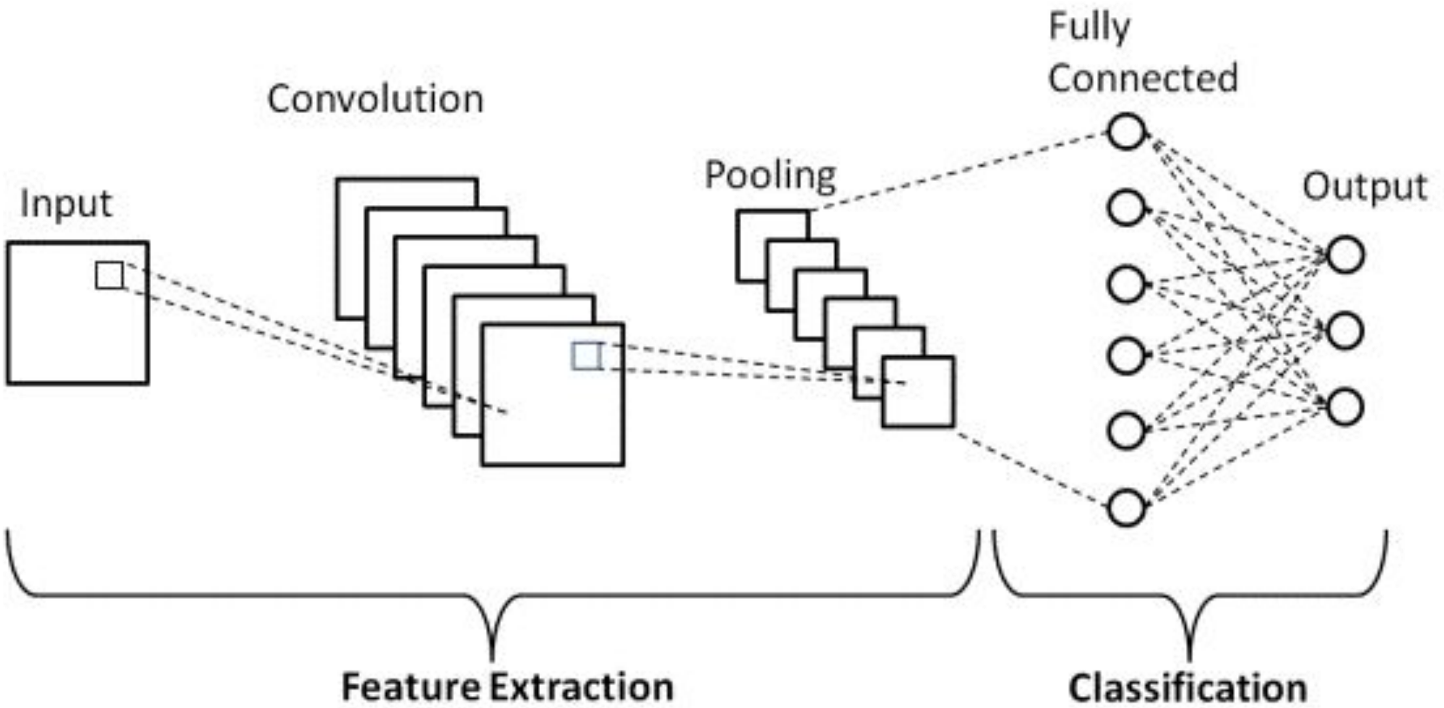
“Learn” the features

MLP


Feature detectors
Feature descriptors
250
249
249
248
242
246
A
44
243
(240, 230, ...)
244
B
42
240
(190, 210, ...)
241
C
40
240
(40, 150, ...)
...
D
..
...
(200, 180, ...)

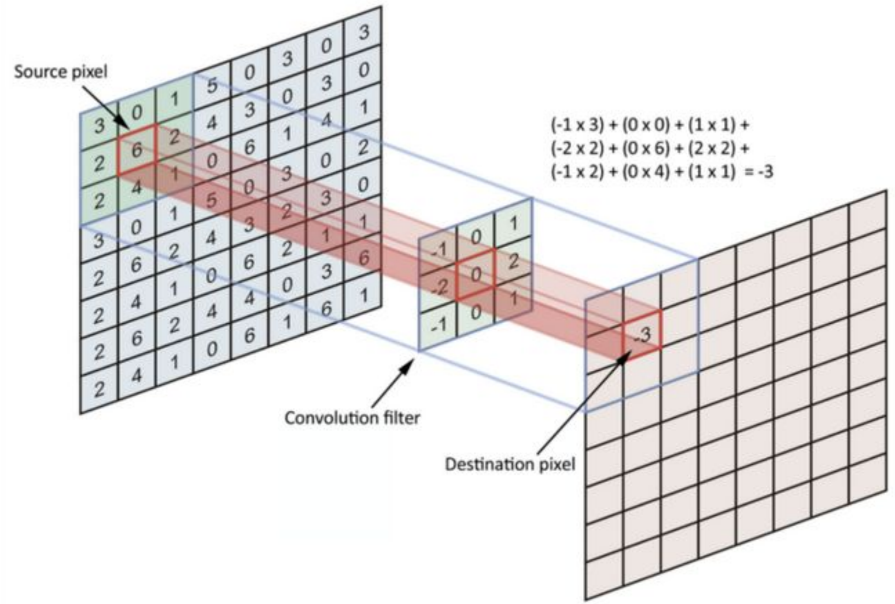

Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Output

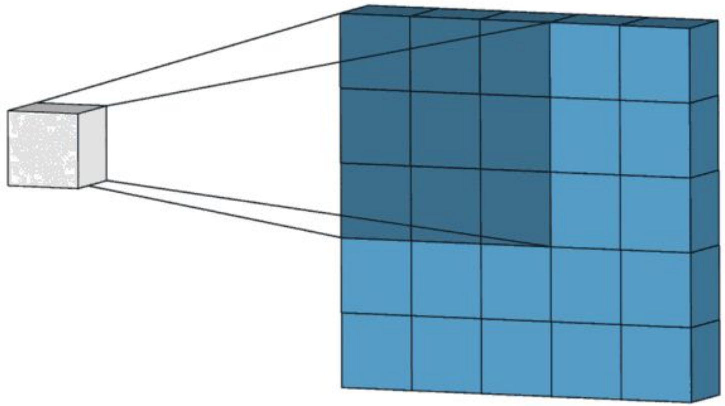
CNN



Convolution
Fully
Connected
Pooling
Input
Output
Feature Extraction
Classification

CNN: convolution layer





Activation function:

- ➢ ReLU: $f(x) = Max(0, x)$

CNN: convolution layer

<u>Image (input):</u>

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

<u>Feature maps (output):</u>

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

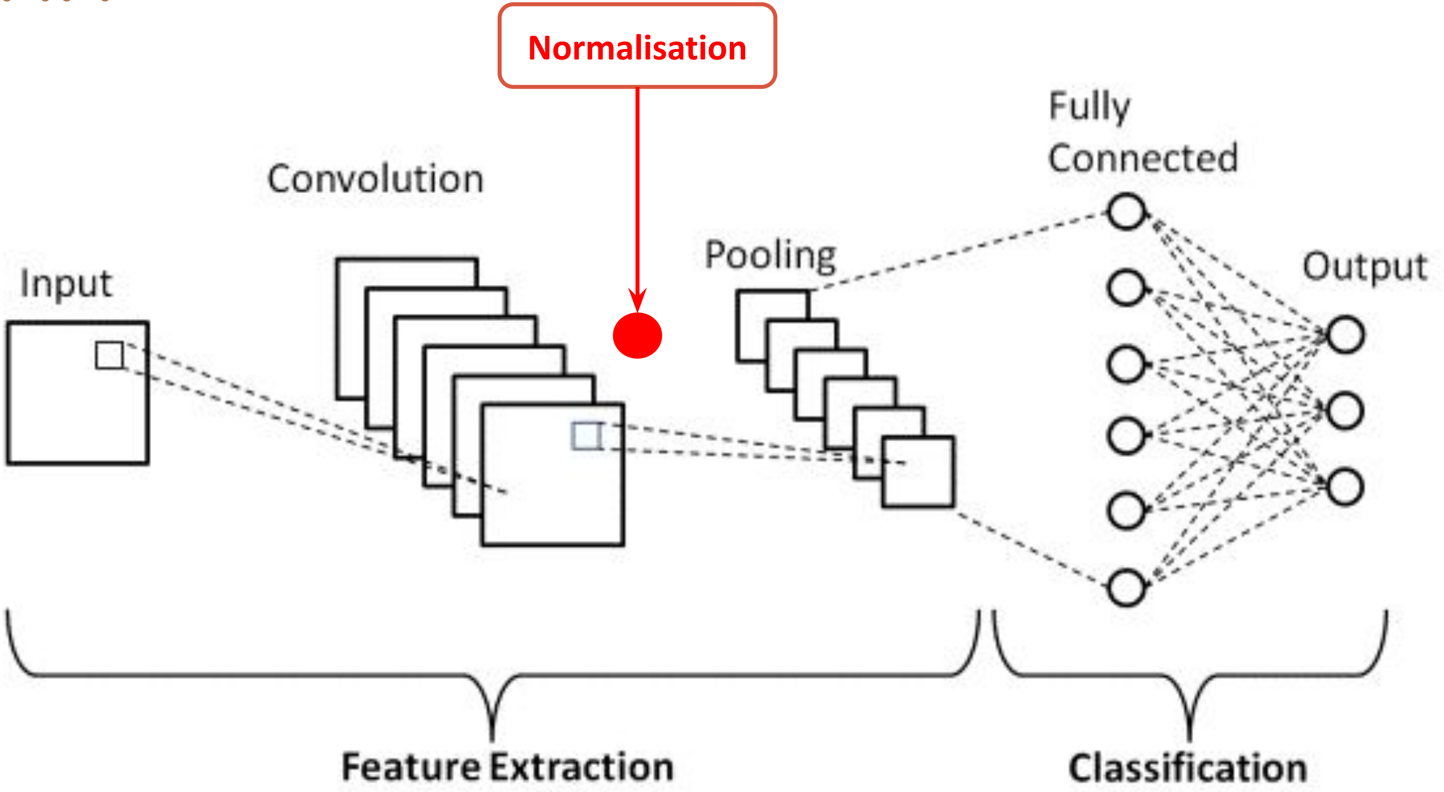
<u>Hyperparameters:</u>

- ➢ K : number of filters
- ➢ F : filter size in pixels (F x F x D)
- ➢ S : sliding step (pixels)
- ➢ P : 0 padding (pixels)
- ➢ Initialization: random / He / Gorot …

<u>Common choice:</u>

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

CNN


Normalisation
Convolution
Fully
Connected
Input
Pooling
Output
Feature Extraction
Classification

CNN: Batch normalization

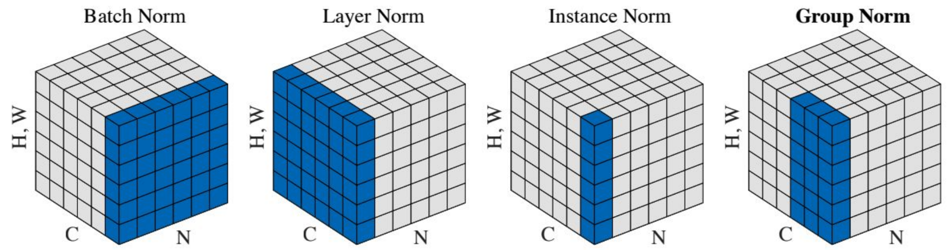
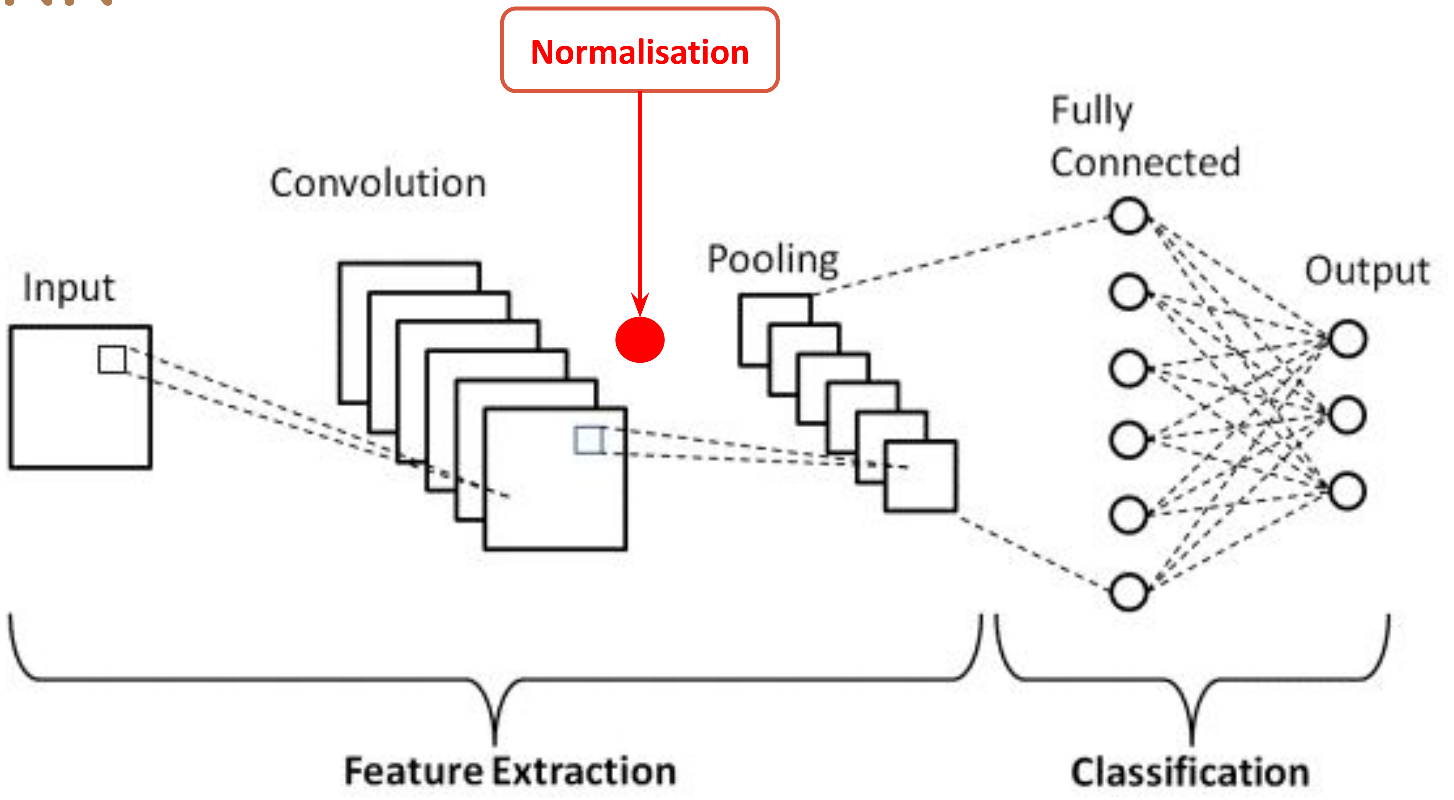


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

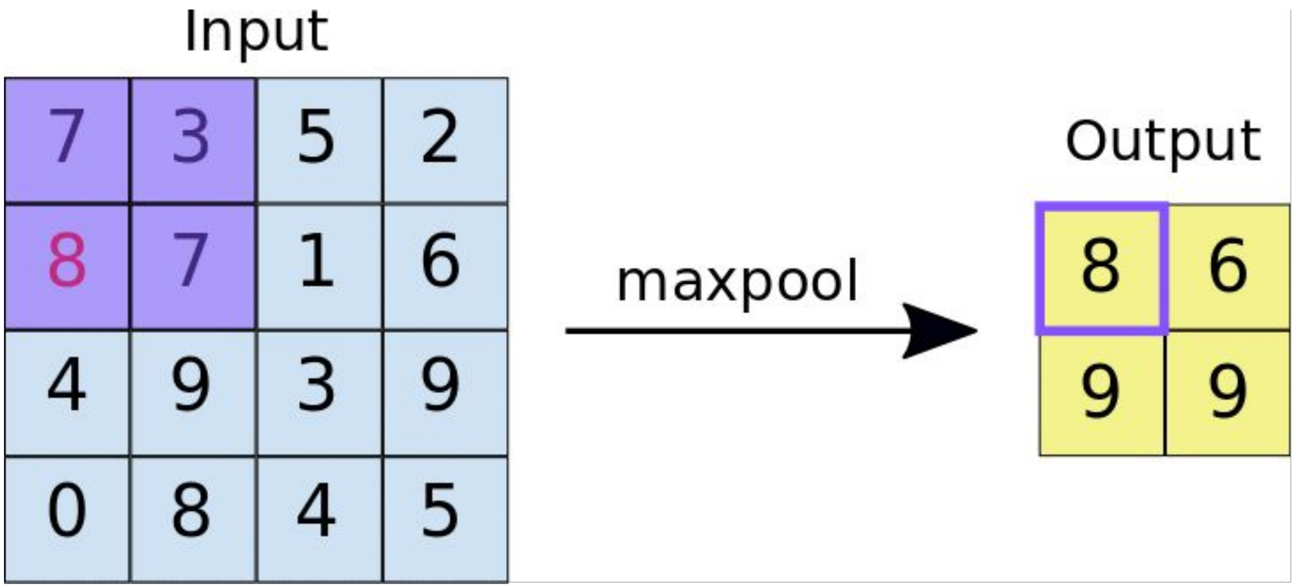
<u>Advantages:</u>

- ➢ Improved Training Speed
- ➢ Better Gradient Flow: smooth the flow
- ➢ Reduced Sensitivity to Initialization
- ➢ Handling Different Batch Sizes

CNN


Normalisation
Convolution
Fully
Connected
Input
Pooling
Output
Feature Extraction
Classification

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

Hyperparameters:

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

Feature maps (output):

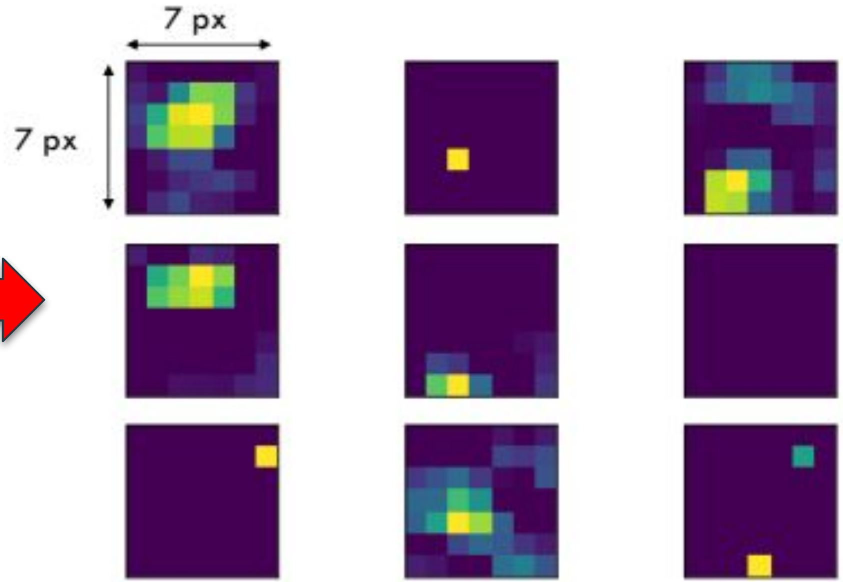
- Wp : (W - F) / S + 1
- Hp : (H - F) / S + 1
- Dp : D

Common choce:

- F = 2 or 3
- S = 2

CNN: Features extraction

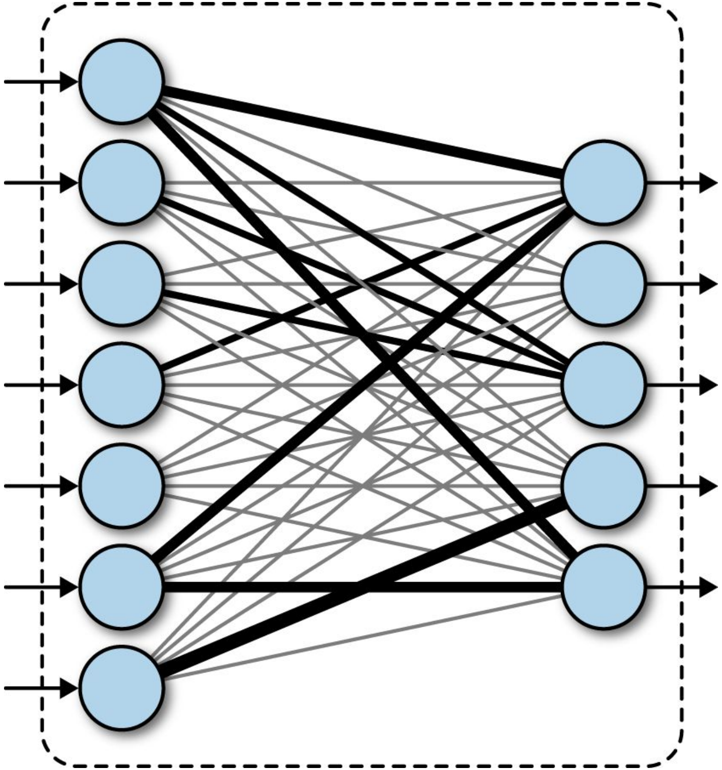
Pooling layer output



CNN: fully-connected layer

Options:

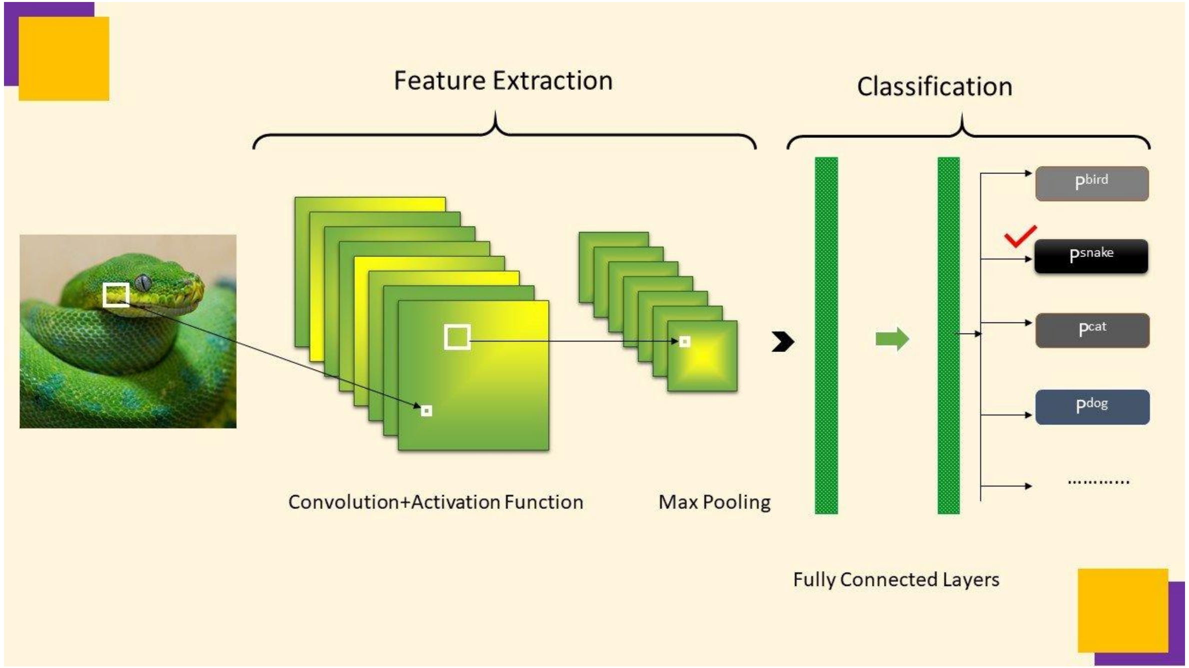
- ➢ Flatten: convert output to 1D vector
- ➢ Number of neurons
- ➢ Dropout: rate
- ➢ Activation:
  - ○ ReLU
  - ○ Classifier 2N: sigmoid
  - ○ Classifier xN: Softmax



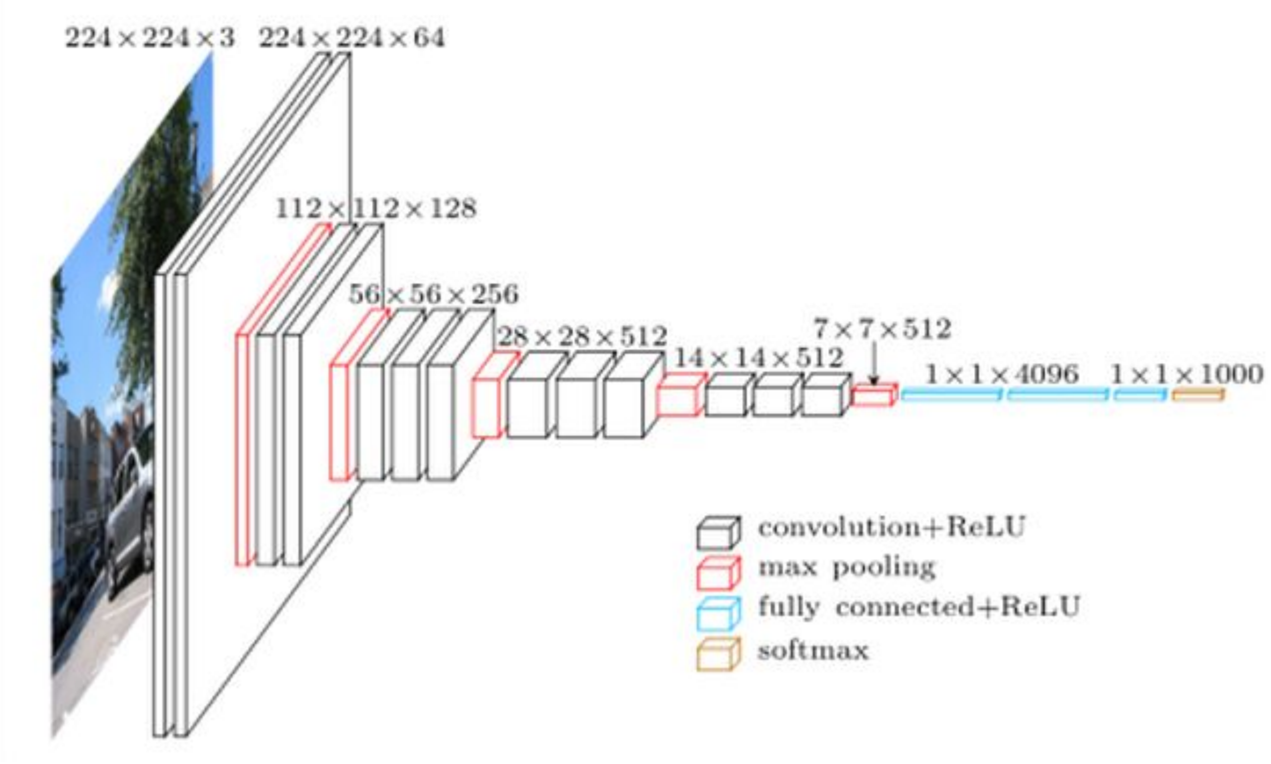
CNN: summary

Key points:

- ➢ Conv layers => extract features
- ➢ Pooling layers => reduce dimensions
- ➢ NN => classify



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

```python
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'))
```

Compilation

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)
```

Which loss ?

- **Classification:**
  - Cross-entropy: Categorical / sparse
- **Object detection:**
  - Intersection over Union (IoU) Loss
  - Smooth L1 Loss => outliers robust
- **Generative Models:**
  - Adversarial Loss: match the generated output distribution to the target distribution.
  - Pixel-wise Mean Squared Error (MSE) Loss
- **Semantic Segmentation:**
  - Pixel-wise Cross-Entropy Loss
  - Dice Loss
- **Instance Segmentation:**
  - Mask-RCNN Loss:
    - Cross-entropy
    - smooth L1
    - binary cross-entropy or Dice loss.

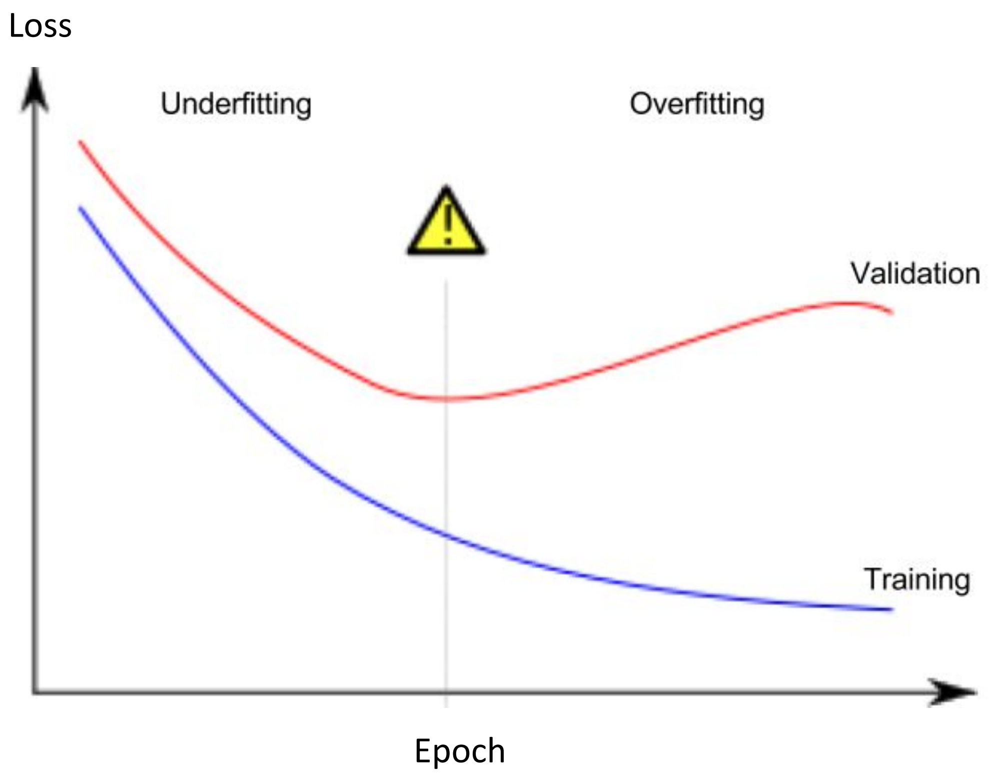
Which optimizer ?

- **SGD:**
  - Small dataset
  - Small CNN architecture
  - No LR scheduler
  - Set LR and Momentum carefully
- **Adam and RMSprop:**
  - LR scheduler
  - LR and Momentum settings have little impact
  - RMSprop slower but more stable
  - RMSprop requires less RAM

Empirical testing required !

Underfitting vs. Overfitting

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



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: strategy

The idea:

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

Strategy:

➢ Pre-trained model initialization

➢ Freezing initial layers

➢ Make the upper layers trainable

➢ Optional: adjusting upper layers: add / remove

➢ Training on the New Task

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

```
########################
### 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)
```

Transfer learning: tools

TensorFlow Hub

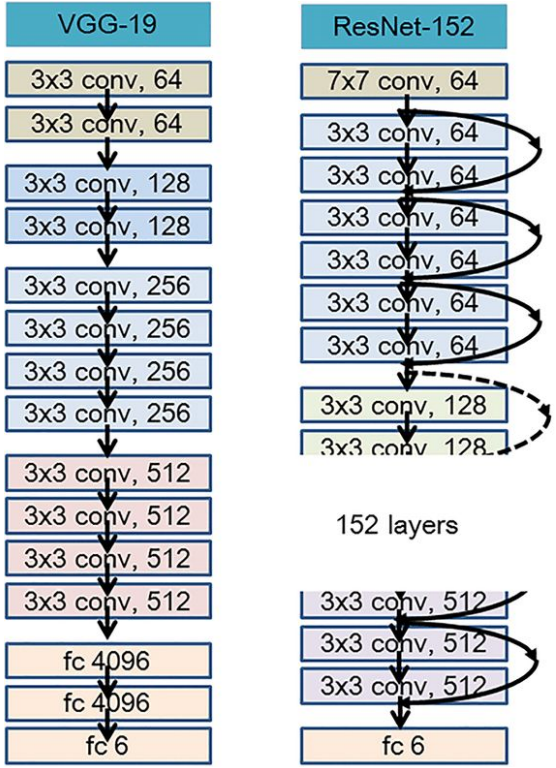
PYTORCH HUB

Hugging Face

Exemples

Architectures

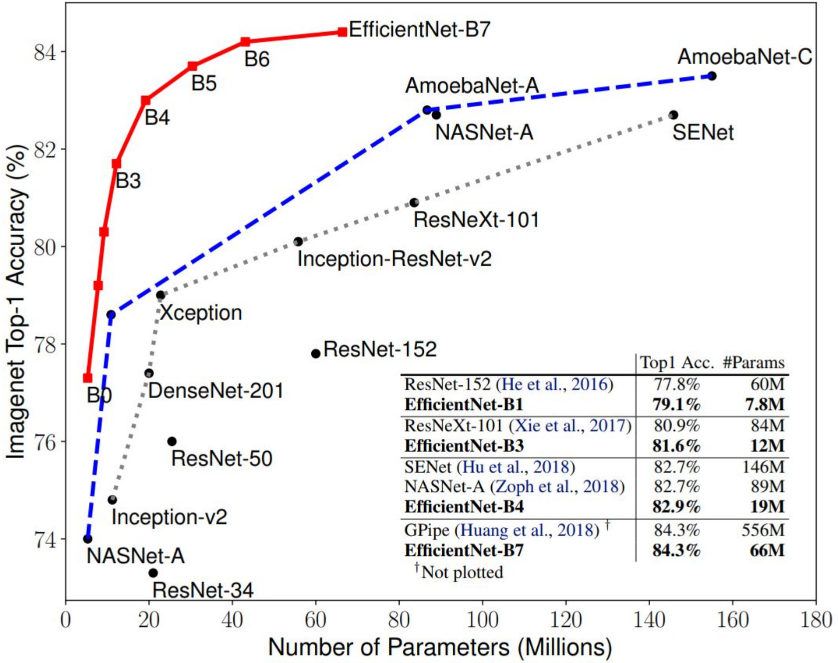
ResNet



Residual connections:

- ➢ Faster training
- ➢ Better performances

EfficiencyNet



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

† Not plotted