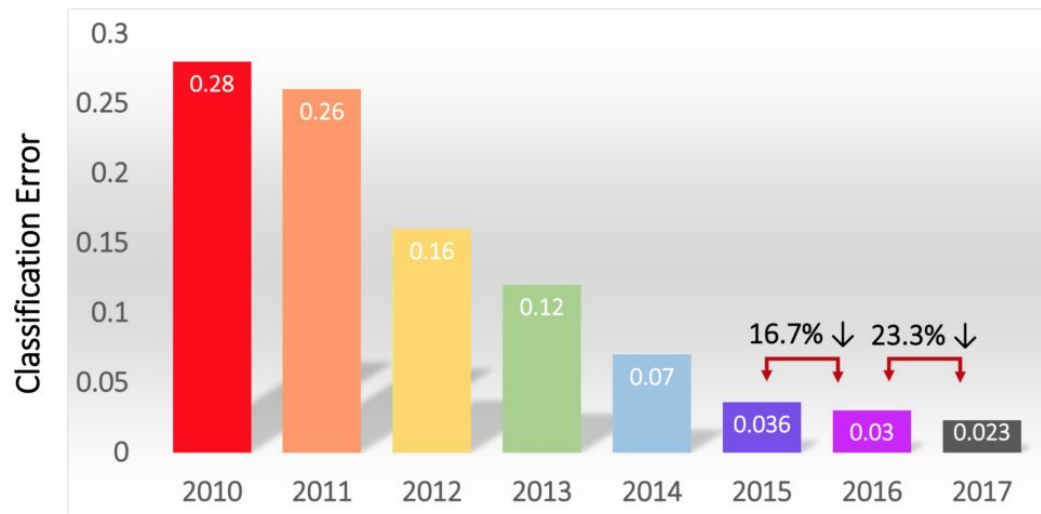


Computer Vision

Modelisation

Plan

Classification Results (CLS)



➤ **Tasks**

➤ **CNN**

➤ **Compilation**

➤ **Transfert learning**

Tasks

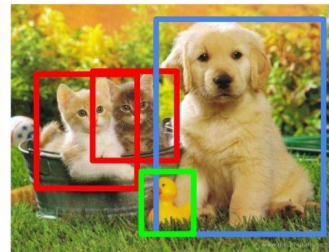
Classification - detection - segmentation - creation



CAT



CAT



CAT, DOG, DUCK



CAT, DOG, DUCK

Single object

Multiple objects

Classification

Particularity:

- N possible outputs with $N = \text{number of classes}$.



CAT

Algorithms

- Mark 1
- SVM
- CNN

Datasets

- ImageNet
- CIFAR 100
- ISIC
- MURA
- DermNet

Models

- LeNet5
- VGG
- GoogleNet / MobileNet
- ResNet

Object detection

Particularity:

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



CAT

Algorithms

- Sliding window
- CNN

Datasets

- ImageNet
- COCO
- PASCAL

Models

- R-CNN
- YOLO
- SSD

Image segmentation

Particularity:

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



CAT, DOG, DUCK

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

This Person Does Not Exist

This Person Does Not Exist

This Person Does Not Existthispersondoesnotexist.com



Algorithms

- CNN

Datasets

- Any...

Models

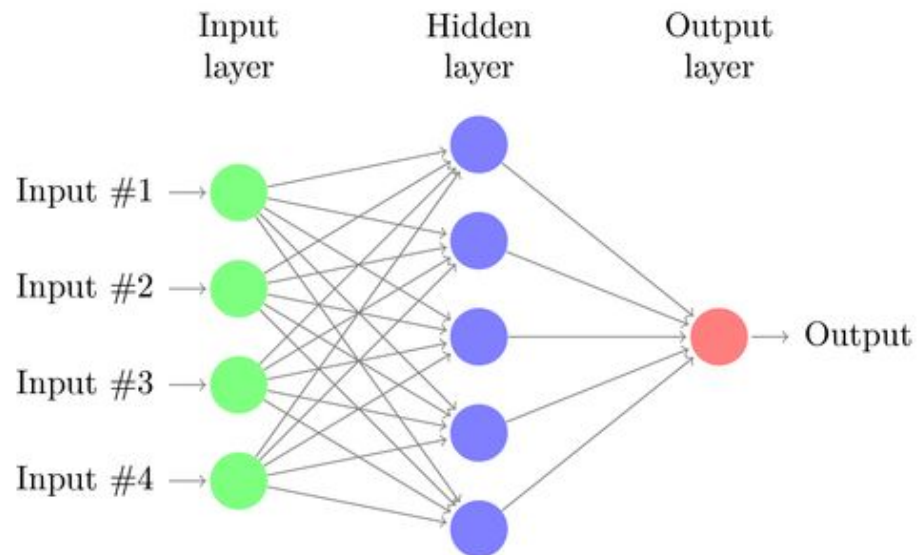
- GANs
- VAEs

CNN

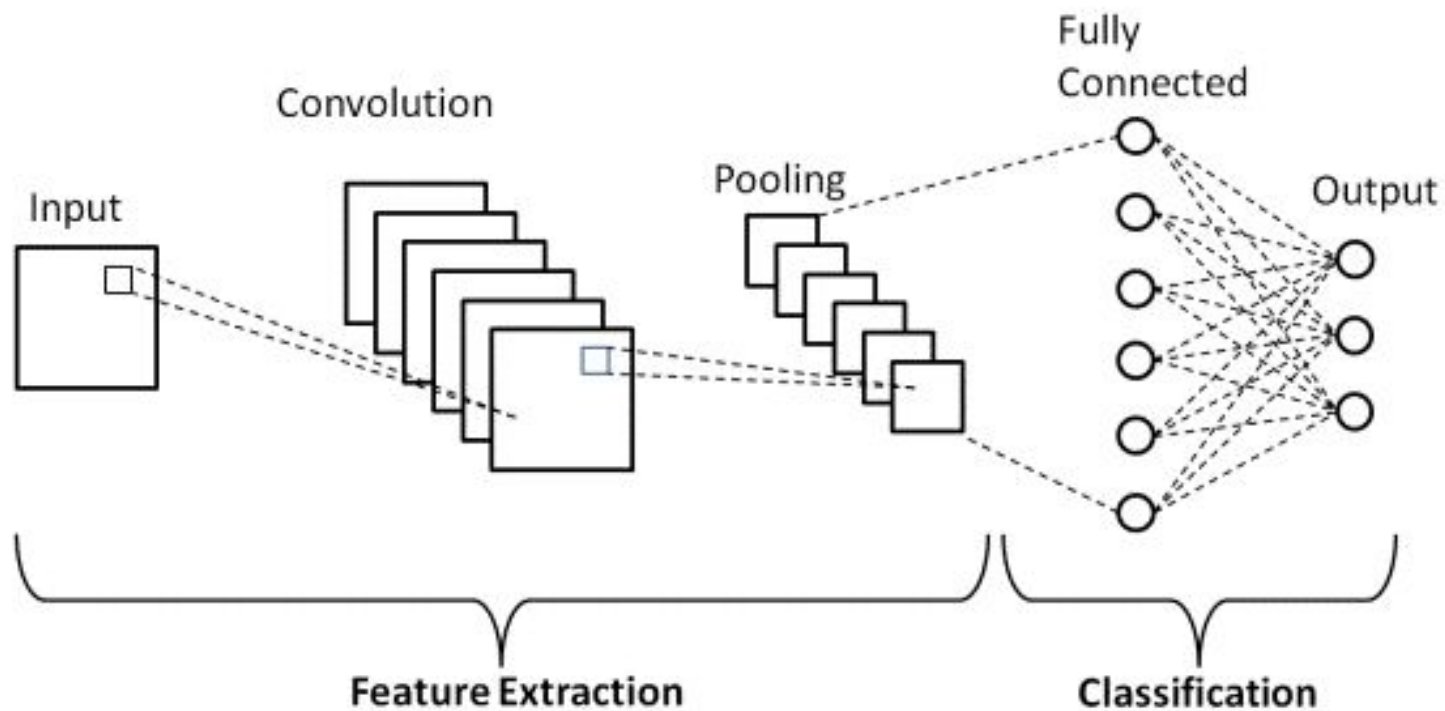
“Learn” the features

MLP

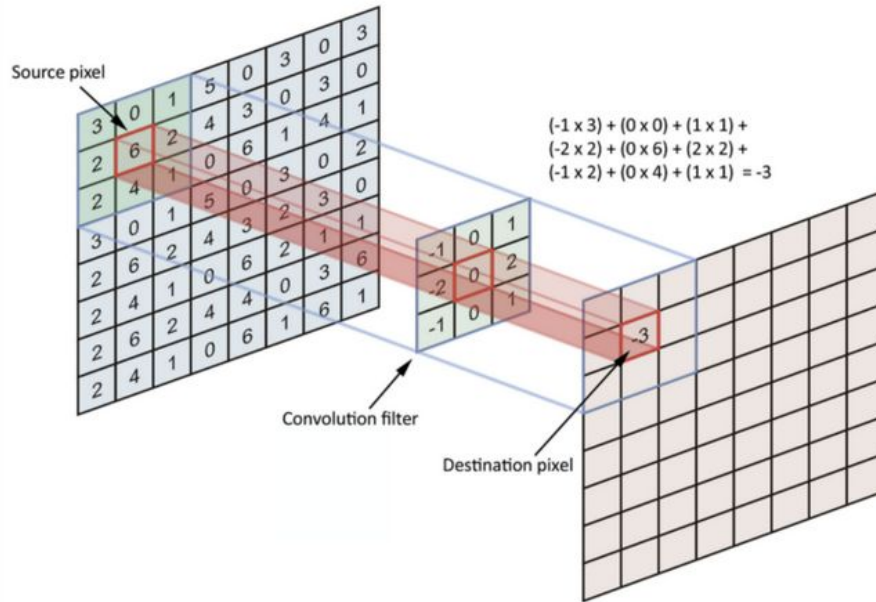
Feature detectors			Feature descriptors	
250	249	249	248	242
246	A	44	243	(240, 230, ...)
244	B	42	240	(190, 210, ...)
241	C	40	240	(40, 150, ...)
...	D	(200, 180, ...)



CNN



CNN: convolution layer



Activation function:

- ReLU: $f(x) = \text{Max}(0, x)$

CNN: convolution layer

Image (input):

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

Hyperparameters:

- 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 ...

Feature maps (output):

- $W_c : (W - F + 2P) / S + 1$
- $H_c : (H - F + 2P) / S + 1$
- $D_c : K$

Common choice:

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

CNN: Batch normalization

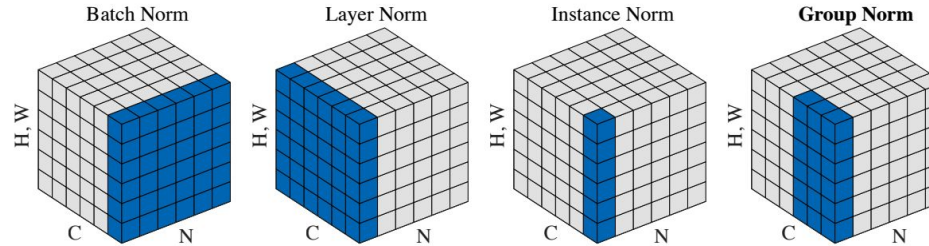
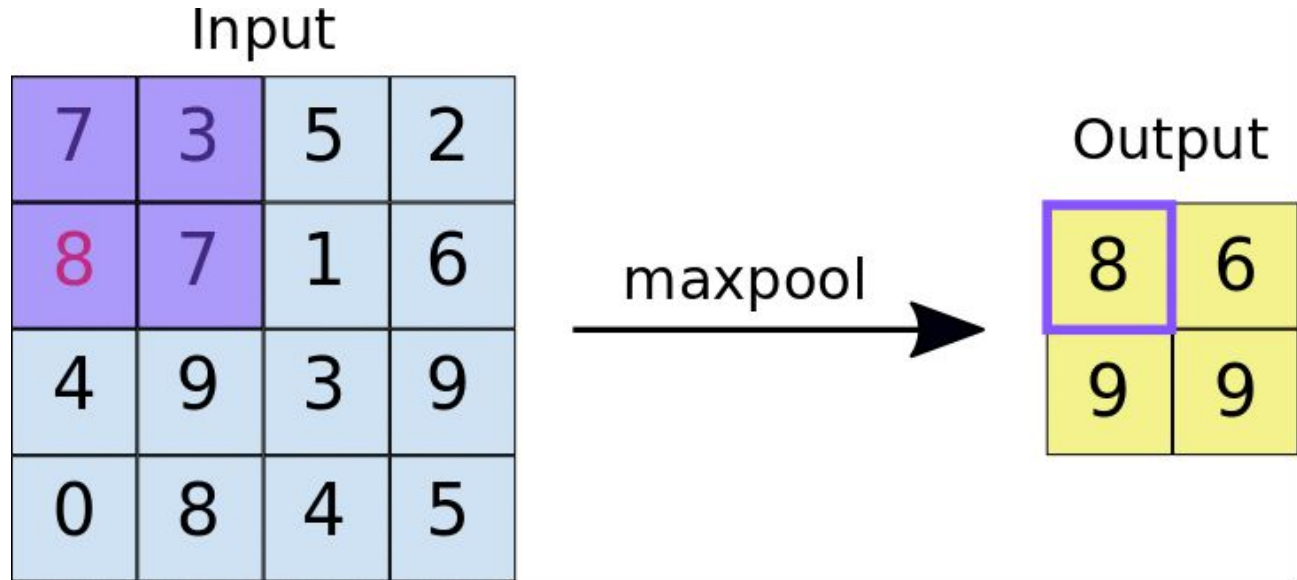


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Advantages:

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

CNN: pooling layer



Advantages:

- Dimension reduction
- Less sensitive to feature position

CNN: pooling layer

Feature maps (input):

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

Hyperparameters:

- F : cell size ($F \times F$ pixels)
- S : pixels between cells

Feature maps (output):

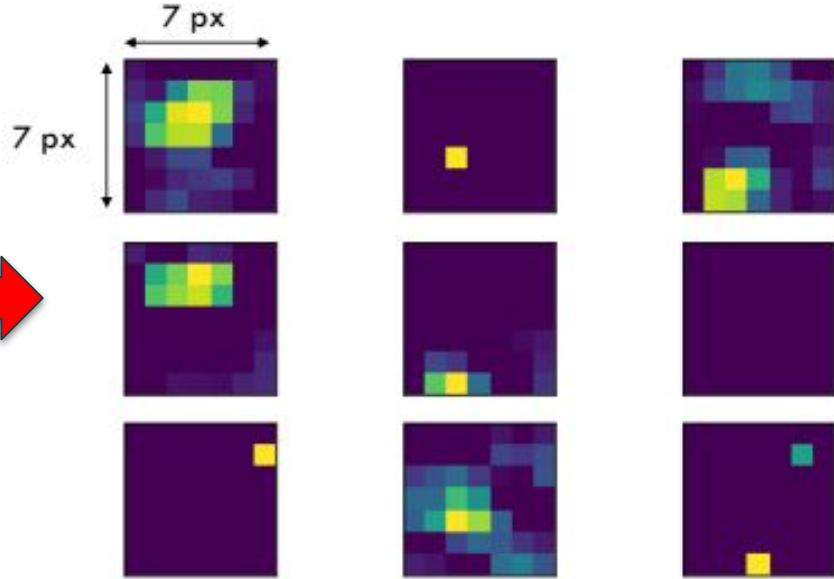
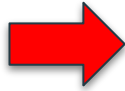
- $W_p : (W - F) / S + 1$
- $H_p : (H - F) / S + 1$
- $D_p : D$

Common choice:

- $F = 2$ or 3
- $S = 2$

CNN: Features extraction

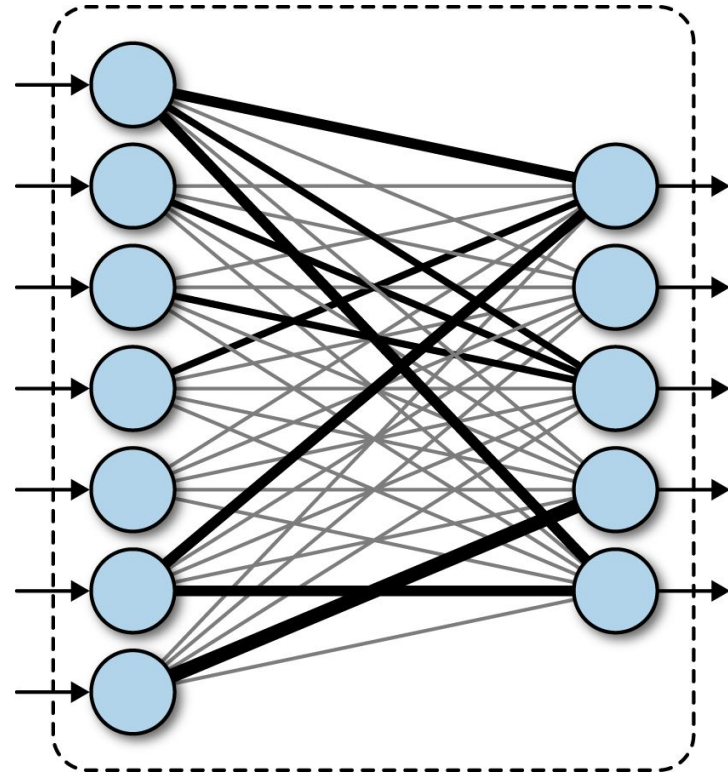
Pooling layer output



CNN: fully-connected layer

Options:

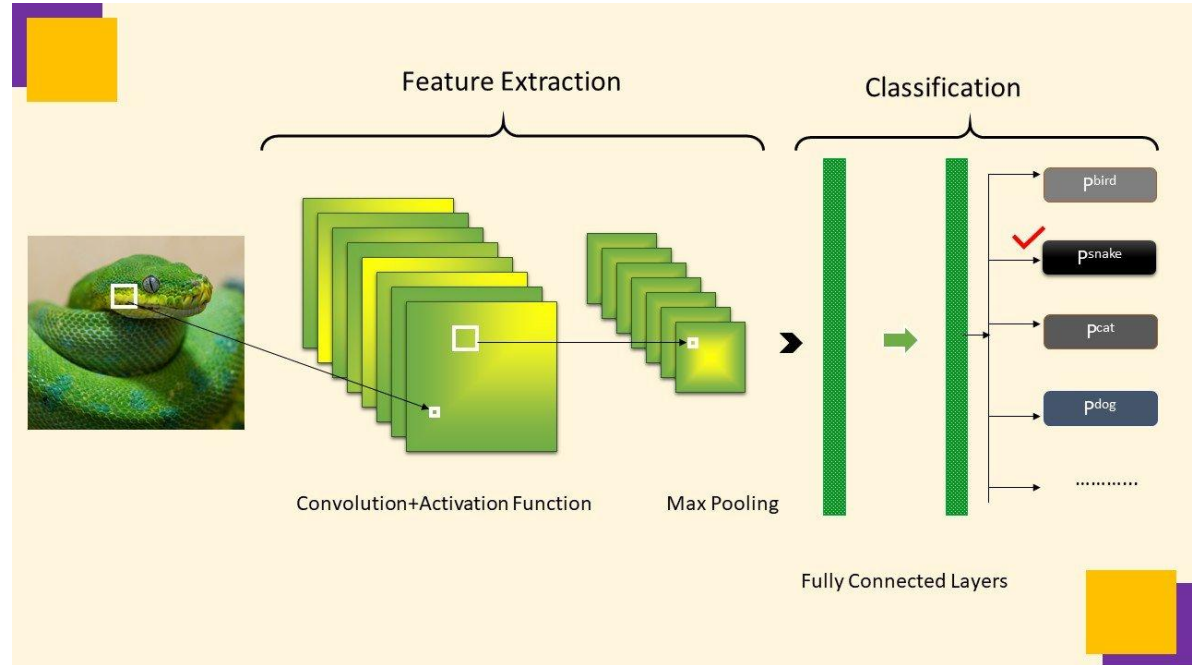
- Flatten: convert output to 1D vector
- Number of neurons
- Dropout: rate
- Activation:
 - ReLU
 - Classifier 2N: sigmoid
 - Classifier xN: Softmax



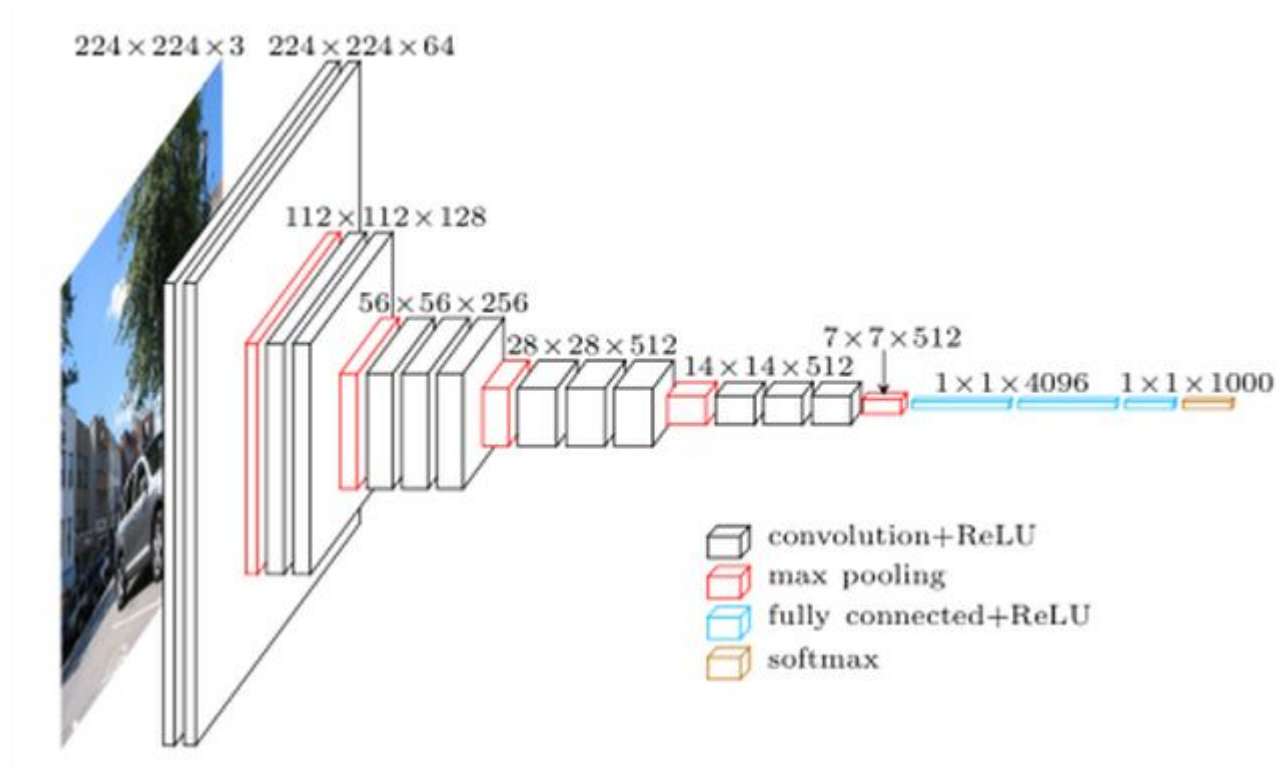
CNN: summary

Key points:

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



CNN: VGG16



CNN: VGG16

```
1 from keras.models import Sequential
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
3
4 # Empty NN
5 my_VGG16 = Sequential()
6
7 #####
8 ### Convolution block ###
9 #####
10
11 # Add the first CNN layer with relu activation
12 my_VGG16.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), padding='same', activation='relu'))
13
14 # Add the second CNN layer with relu activation
15 my_VGG16.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
16
17 # Add the first pooling layer
18 my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
19
20 # Repeat the previous steps as often as needed...
21
22 #####
23 ### Fully-connected block ###
24 #####
25
26 # Convert 3D matrix to 1D vector
27 my_VGG16.add(Flatten())
28
29 # Add first fully-connected layer with relu activation
30 my_VGG16.add(Dense(4096, activation='relu'))
31
32 # Add second fully-connected layer with relu activation
33 my_VGG16.add(Dense(4096, activation='relu'))
34
35 # Add last fully-connected layer which is the classifier
36 my_VGG16.add(Dense(1000, activation='softmax'))
37
```

Compilation

The suspense is intense !

Hyperparameters

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

```
67 #####
68 #####
69 ### Transfer learning ###
70 #####
71
72 from keras import Model
73
74 # Load the VGG16 trained on imagenet images without the fully-connected layer
75 model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
76
77 # Get the output of VGG16
78 x = model.output
79
80 # Add your new classifier
81 predictions = Dense(10, activation='softmax')(x)
82
83 # Design your new model
84 new_model = Model(inputs=model.input, outputs=predictions)
85
86 # Strat n°1:
87 for layer in new_model.layers:
88     layer.trainable = True
89
90 # Strat n°2:
91 for layer in new_model.layers:
92     layer.trainable = False
93
94 # Strat n°3: let's not train the first 5 layers (the largest)
95 for layer in new_model.layers[:5]:
96     layer.trainable = False
97
98 # Compile the new model
99 new_model.compile(loss="categorical_crossentropy", optimizer=optimizers.SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])
100
101 # Train it
102 model_info = new_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=2)
103
104
```

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:**

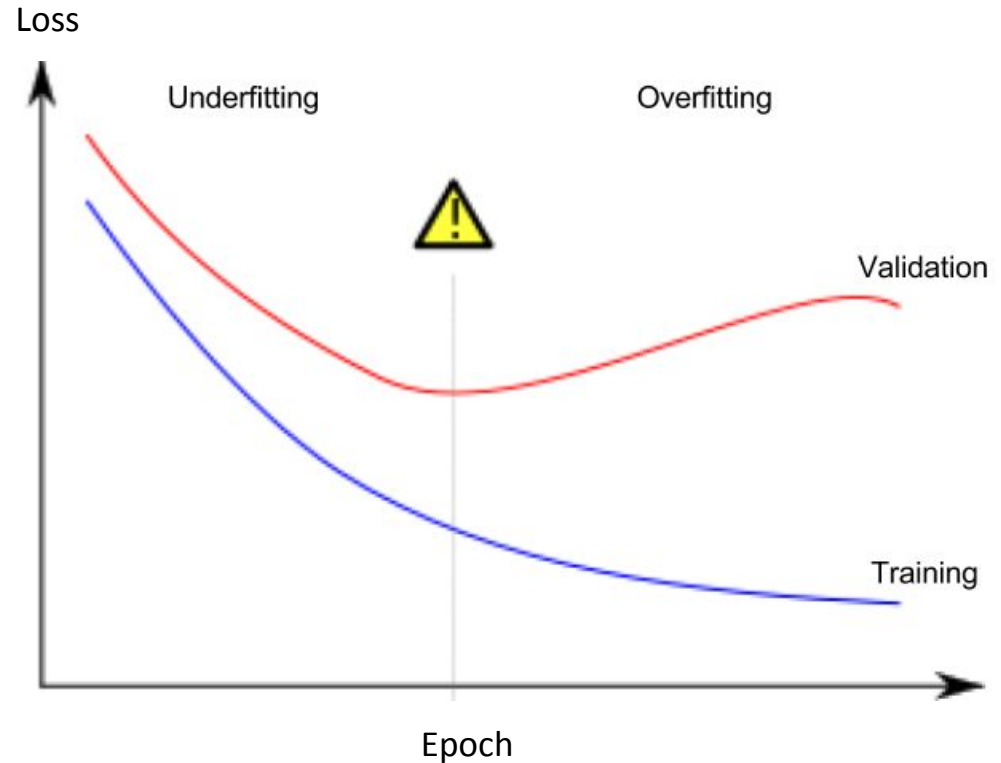
- LR scheduler
- LR and Momentum settings have little impact
- RMSprop slower but more stable
- RMSprop requires less RAM



**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
 - $\text{weights} = F \times F \times K + K$
- Fully-connected:
 - W : width of input
 - H : height of input
 - K : number of filters
 - N : number of neurons
 - $\text{Weights} = W \times H \times K \times N + N$

**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

```
38
39 #####
40 ### Pre-trained exemple ###
41 #####
42
43 from keras.applications.vgg16 import VGG16
44
45 # Get VGG16 available in keras
46 model = VGG16()
47
48 from keras.preprocessing.image import load_img, img_to_array
49 from keras.applications.vgg16 import preprocess_input
50
51 # Load image and reshape image to match VGG16 dimensions expectation
52 img = load_img('cat.jpg', target_size=(224, 224))
53
54 # Convert to numpy array because keras processes images as such
55 img = img_to_array(img)
56
57 # A CNN expects (in general) a collection of images.
58 # Reshape the array to add the number of images
59 img = img.reshape((1, img.shape[0], img.shape[1], img.shape[2]))
60
61 # Preprocess images the same way the images used to train VGG16 were preprocessed.
62 img = preprocess_input(img)
63
64 # Predict the class (or label) of this image
65 y = model.predict(img)
66
```

Transfer learning: exemple

```
67 #####
68 ### Transfer learning ###
69 #####
70
71
72 from keras import Model
73
74 # Load the VGG16 trained on imagenet images without the fully-connected layer
75 model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
76
77 # Get the output of VGG16
78 x = model.output
79
80 # Add your new classifier
81 predictions = Dense(10, activation='softmax')(x)
82
83 # Design your new model
84 new_model = Model(inputs=model.input, outputs=predictions)
85
86 # Strat n°1:
87 for layer in new_model.layers:
88     layer.trainable = True
89
90 # Strat n°2:
91 for layer in new_model.layers:
92     layer.trainable = False
93
94 # Strat n°3: let's not train the first 5 layers (the largest)
95 for layer in new_model.layers[:5]:
96     layer.trainable = False
97
98 # Compile the new model
99 new_model.compile(loss="categorical_crossentropy", optimizer=optimizers.SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])
100
101 # Train it
102 model_info = new_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=2)
103
104
```

Transfer learning: tools



TensorFlow Hub

PYTORCH
HUB



Hugging Face

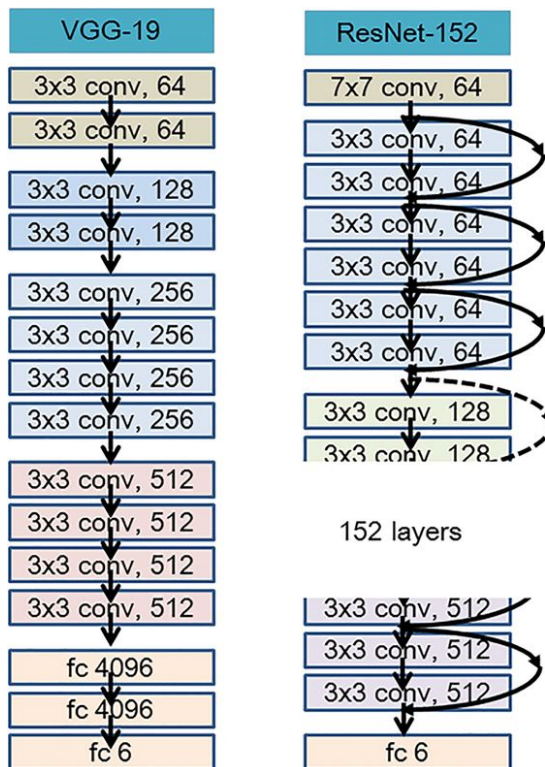
Examples

Architectures

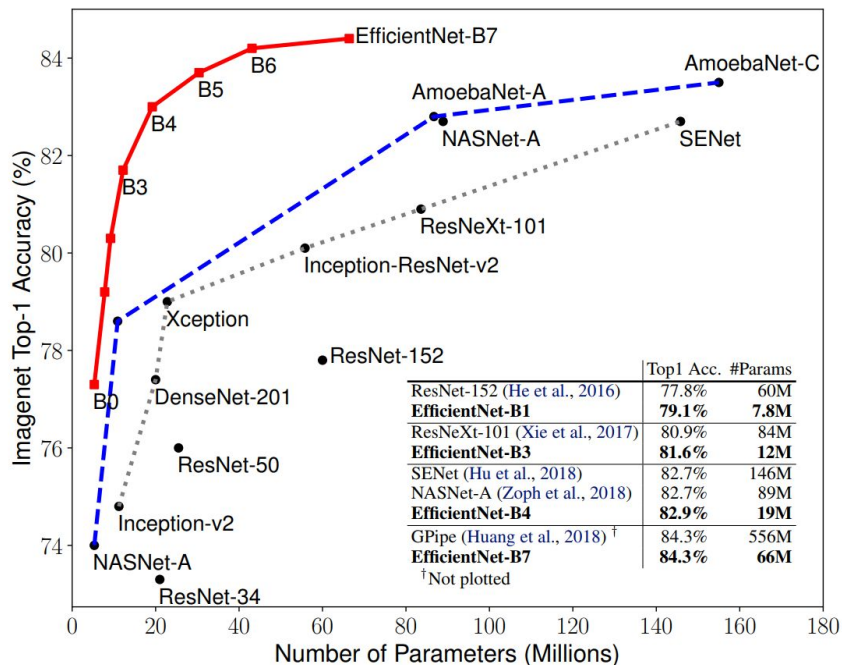
ResNet

Residual connections:

- Faster training
- Better performances



EfficiencyNet



EfficientNet Architecture

