_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Advanced methods of artificial vision

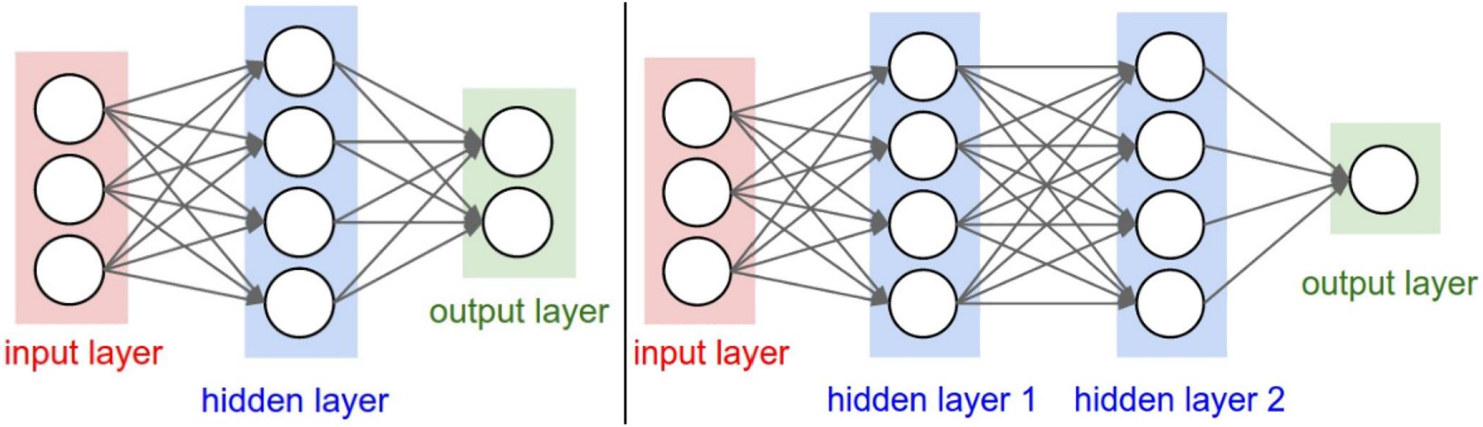
Unit 2: CNN-based feature extraction

Contents

Contents

Neural Networks: Structure

A neural network (NN) is a set of layers composed of neurons.

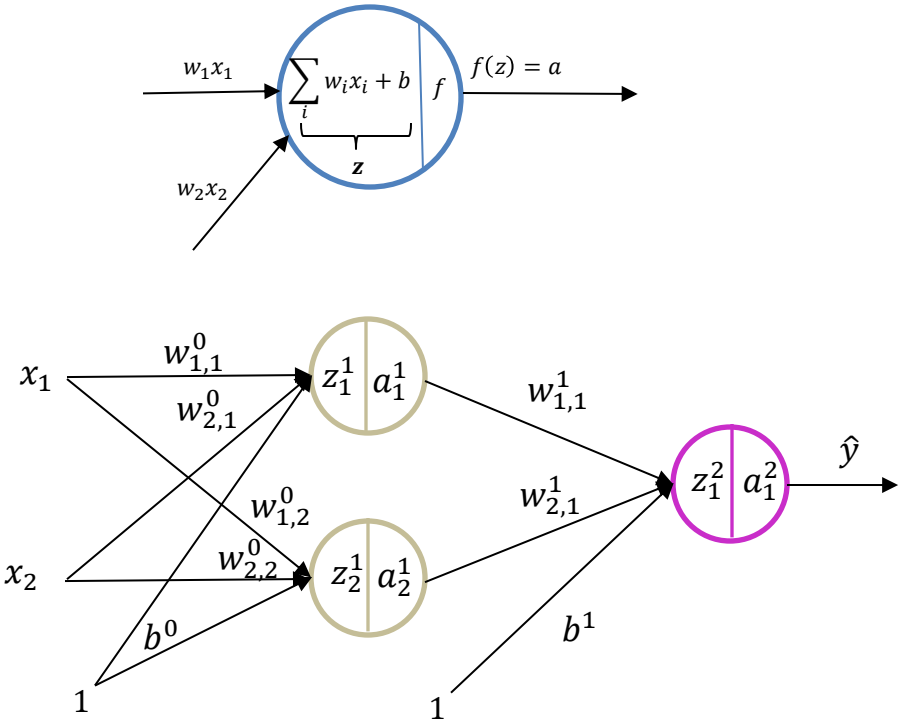


NN of 2 layers

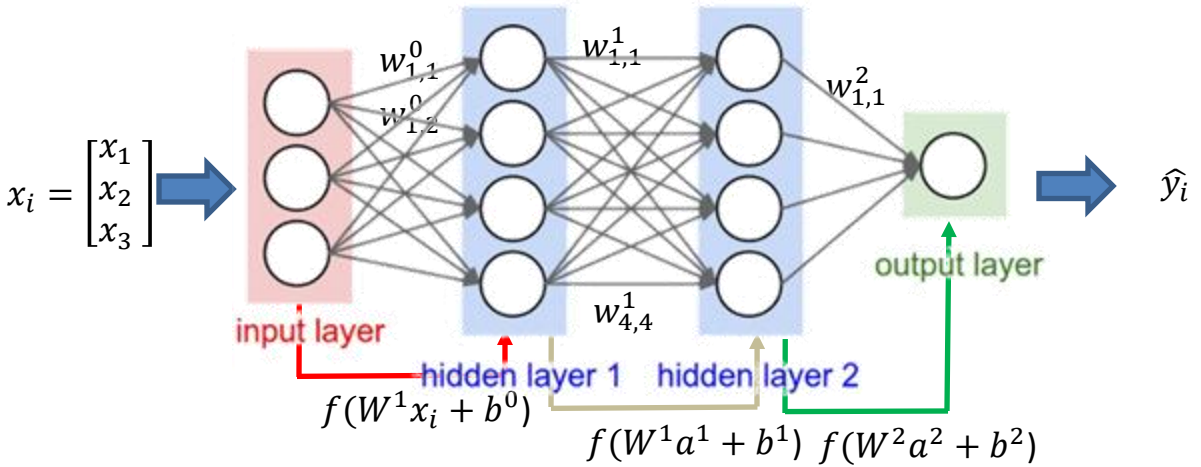
NN of 3 layers

fully connected layers: Denses

Neural Networks: Structure



Neural Networks: Structure

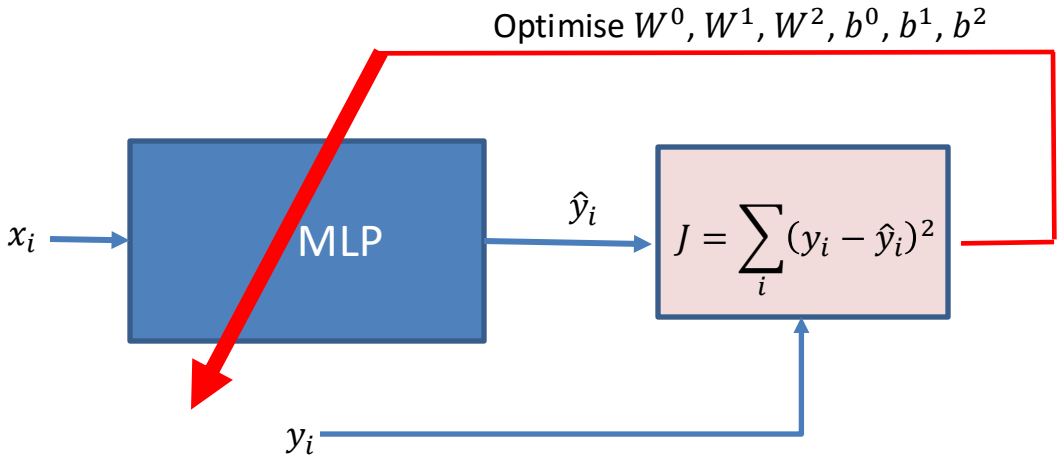


$$W^0 = \begin{bmatrix} w^0_{1,1} & w^0_{2,1} & w^0_{3,1} \\ w^0_{1,2} & w^0_{2,2} & w^0_{3,2} \\ w^0_{1,3} & w^0_{2,3} & w^0_{3,3} \\ w^0_{1,4} & w^0_{2,4} & w^0_{3,4} \end{bmatrix}_{D_1 \times D}$$

$$W^1 = [\quad]_{D_2 \times D_1}$$

$$W^2 = [\quad]_{K \times D_2}$$

Optimization

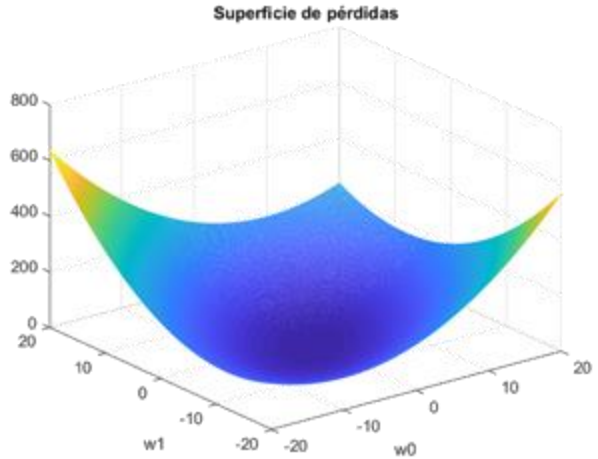


$$J = -\frac{1}{m}\sum_{i}^{m} y_i log(\hat{p}_i) + (1 - y_i) log(1 - \hat{p}_i)$$

Optimization

$$J = \sum_i (y_i - \hat{y}_i)^2 \longrightarrow argmin_{W,b}\{J\}$$

$$J = 2 + 0.5 \times (w_0^2 + w_1^2) - 0.5 \times w_0 \times w_1 + 1.7 \times w_1$$



$$\nabla J = 0 \longrightarrow \nabla J = \left[\frac{\partial J}{\partial w_0} \quad \frac{\partial J}{\partial w_1}\right] = 0$$

To solve this system

Optimization: Gradient Descent

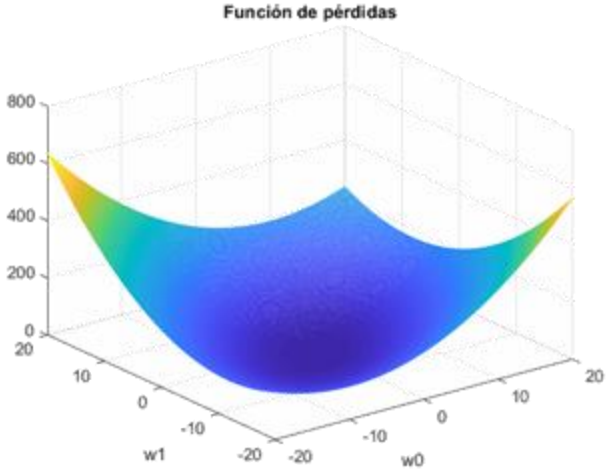
Iterative process:

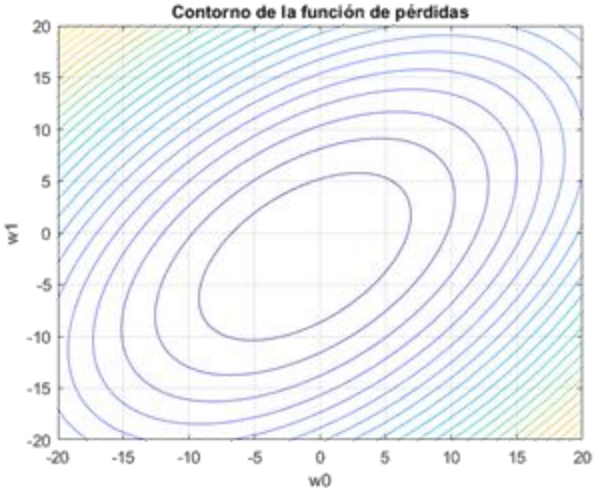
$$w_0(t+1) = w_0(t) - \eta \frac{\partial J}{\partial w_0}$$

$$w_1(t+1) = w_1(t) - \eta \frac{\partial J}{\partial w_1}$$

η: learning rate

- **High values:** fast learning but be careful with convergence
- **Valores bajos:** slow learning





Gradient descent variants

- Gradient Descent: original (vanilla version)
  - For each training set data:
    - Fordward pass
    - Error calculation and accumulation
  - Update the weights (backpropagation)
- Stocasthic Gradient Descent (SGD)
  - For each training set data:
    - Fordward pass
    - Error calcutlation and accumulation
    - Update the weights (backpropagation)
- Minibacth Gradient Descent (SGD)
  - Divide the training set in batches
  - For each batch
    - For each data from the batch
      - Fordward pass
      - Error calculation and accumulation
    - Update the weights (backpropagation)

The weights are updated only once in the training set

Consume a lot of resources but it is more precise

The weights are updated for each data in the training set

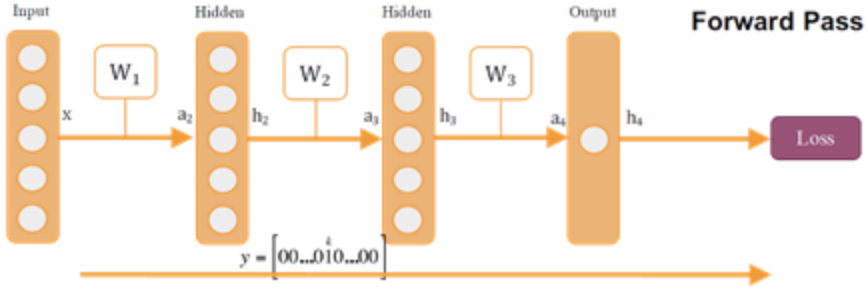
The weights are updated for each batch in the training set

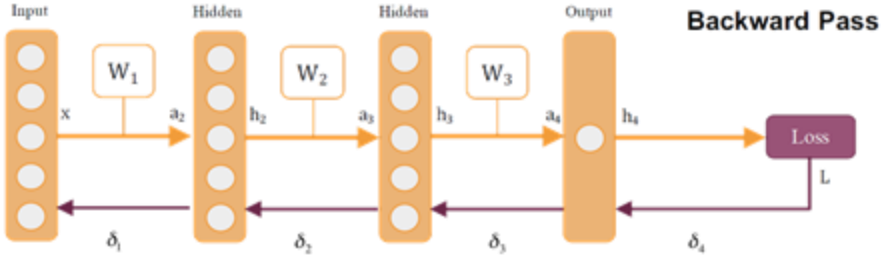
Optimal with vectorised versions of the algorithm

Hyperparameter to optimise, the batch size: for example in CNNs it is usually 256 for a training set of 1.2 million.

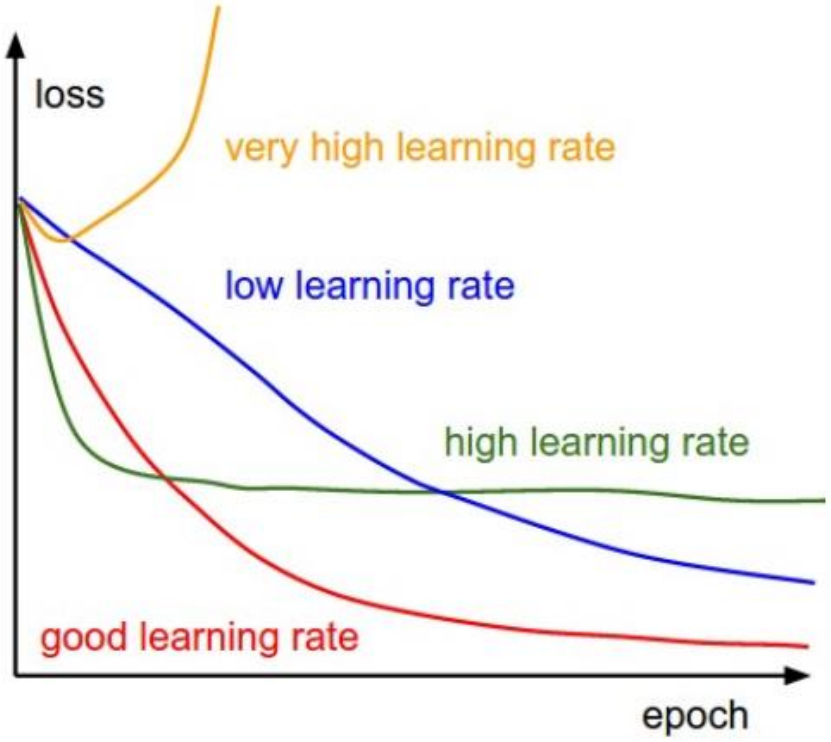
Forward-Backward propagation

Training data

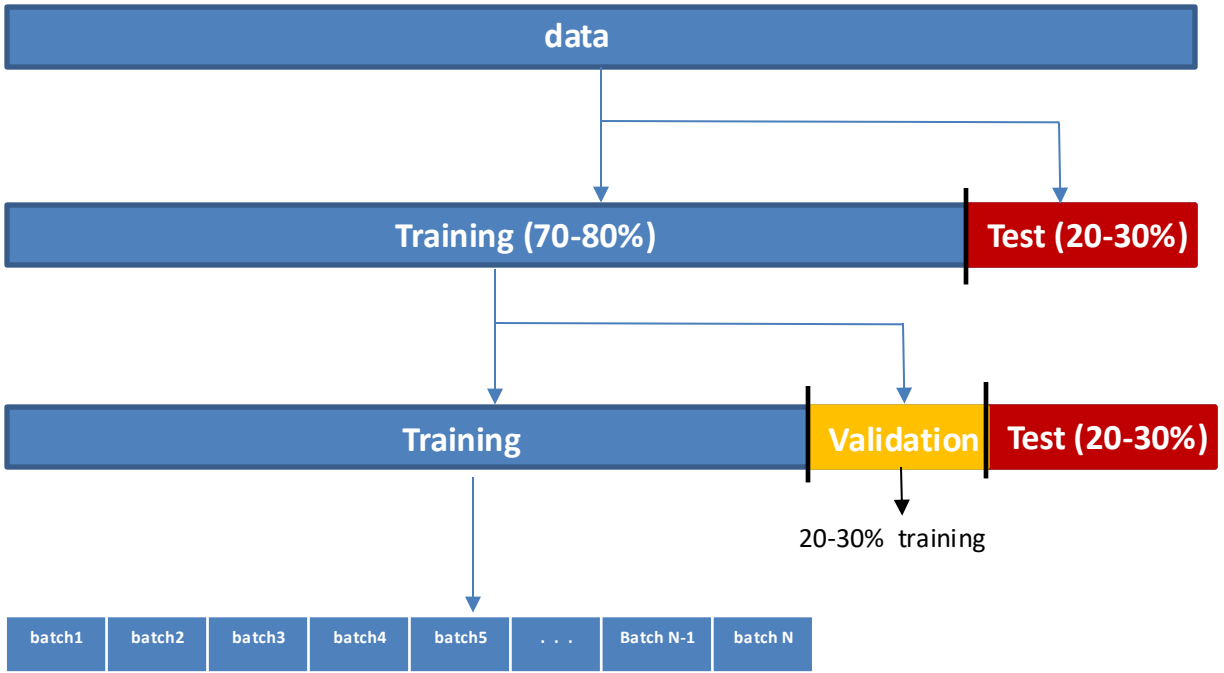




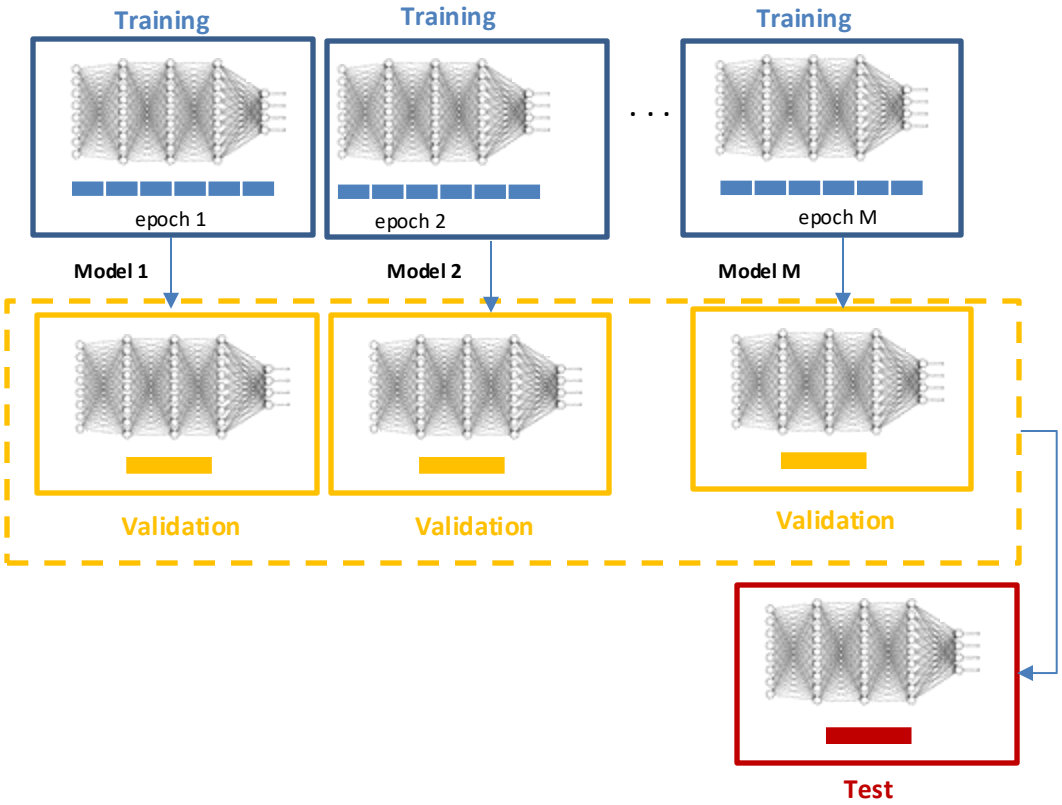
Effect of *learning rate*

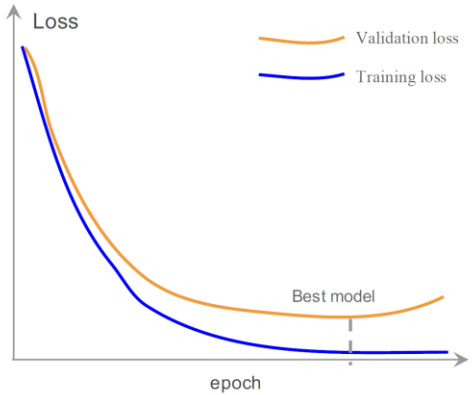


Data partitioning



Training process

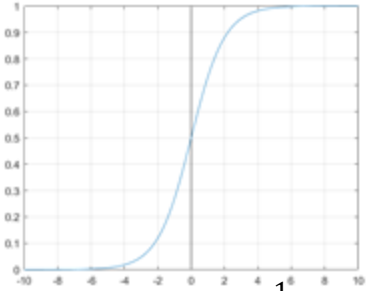




Model with highest accuracy or lowest validation loss

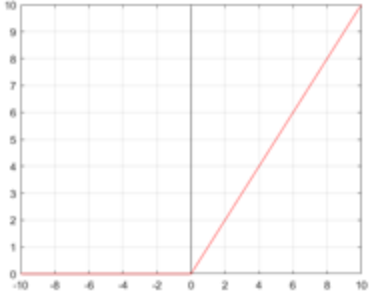
Activation functions

Sigmoid



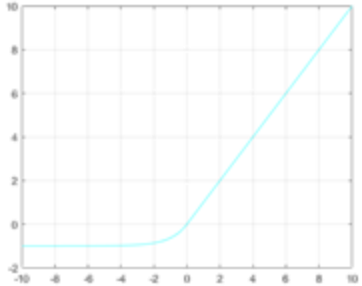
$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Relu (Rectified Linear Unit)



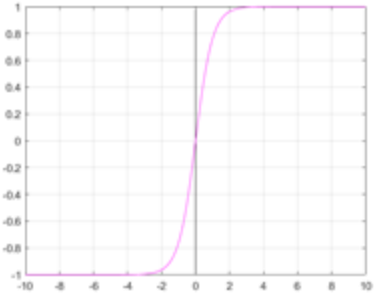
$$f(x) = \max(0, x)$$

Elu (Exponential Linear Unit) (α=1)



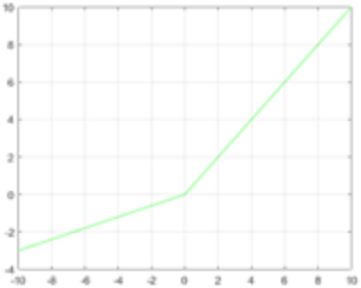
$$f(x) = \alpha * (e^x - 1)\,(x < 0) + x(x \geq 0)$$

Hyperbolic tangent



$$f(x) = tanh(x) = 2\sigma(2x) - 1$$

Leaky Relu (α=0.3)



$$f(x) = \alpha * x\,(x < 0) + x(x \geq 0)$$

Optimizers

Year	Optimizer	Description	Formula
1950	Gradient Descent (GD)	Updates the weights by moving them in the direction opposite to the gradient of the loss function. Requires a fixed learning rate.	$w = w - \eta \cdot \nabla L(w)$ where η is the learning rate, w are the weights, and $\nabla L(w)$ is the gradient of the loss function with respect to the weights.
1983	Momentum	Accumulates the past gradients to dampen oscillations, helping speed up training and avoiding local minima.	$v_t = \gamma v_{t-1} + \eta \cdot \nabla L(w)$ $w = w - v_t$ where γ is the momentum factor and v_t is the accumulated velocity.
1983	Nesterov Accelerated Gradient (NAG)	Similar to momentum but computes the gradient in the "future" direction, making more precise adjustments to the weights.	$v_t = \gamma v_{t-1} + \eta \cdot \nabla L(w - \gamma v_{t-1})$ $w = w - v_t$ The gradient is computed at $w - \gamma v_{t-1}$, different from standard momentum.
2011	AdaGrad (Adaptive Gradient Algorithm)	Adapts the learning rate for each parameter based on the past gradient magnitudes, allowing rare parameters to be updated faster.	$r_t = r_{t-1} + \nabla L(w_t)^2$ $w_t = w_{t-1} - \frac{\eta}{\sqrt{r_t + \epsilon}} \cdot \nabla L(w_t)$ where r_t is the accumulated gradient sum and ϵ is a small value to prevent division by zero.

- η: Learning rate
- w_t: Weights at time step t
- $\nabla L(w)$: Gradient of the loss function L with respect to the weights
- γ: Momentum factor
- r_t: Sum of past gradients (AdaGrad)

Year	Optimizer	Description	Formula
2012	**RMSProp (Root Mean Square Propagation)**	A modification of AdaGrad, using an exponentially decaying average of squared gradients to avoid AdaGrad's diminishing learning rate problem.	$r_t = \beta r_{t-1} + (1-\beta)\nabla L(w_t)^2$ $w_t = w_{t-1} - \frac{\eta}{\sqrt{r_t+\epsilon}} \cdot \nabla L(w_t)$ β is the decay rate for the moving average of squared gradients.
2012	**AdaDelta**	An improvement on AdaGrad, designed to prevent the rapid decay of the learning rate by considering only a window of recent gradients.	$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$ $\Delta w_t = -\frac{\eta}{\sqrt{E[\Delta w^2]_t+\epsilon}} g_t$ ρ controls the window of the moving average.
2014	**Adam (Adaptive Moment Estimation)**	Combines the advantages of both AdaGrad and RMSProp by using a running average of both the gradients and their squared magnitudes.	$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L(w_t)$ $v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla L(w_t)^2$ $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$ $w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}} \cdot \hat{m}_t$
2016	**Nadam (Nesterov Adam)**	Combines Nesterov Accelerated Gradient (NAG) with Adam, applying the momentum step before calculating the gradient.	Similar to Adam, but incorporates NAG-style updates: $w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}}(\hat{m}_t + \gamma \cdot \hat{m}_{t+1})$
2018	**AMSGrad**	A variant of Adam that fixes convergence issues in some cases by maintaining the maximum of past squared gradients rather than the average.	Same as Adam, but updates the second moment v_t with the maximum: $v_t = \max(v_{t-1}, v_t)$.

- η: Learning rate
- w_t: Weights at time step t
- $\nabla L(w)$: Gradient of the loss function L with respect to the weights
- γ: Momentum factor
- r_t: Sum of past gradients (AdaGrad)
- ϵ: Small value to prevent division by zero
- β_1, β_2: Exponential decay rates for moment estimates (Adam)
- m_t: First moment (mean of gradients, Adam)
- v_t: Second moment (mean of squared gradients, Adam)
- $\hat{m}_t, \hat{v}_t$: Bias-corrected first and second moments (Adam)

NN: learning rate optimization

Vary the learning rate throughout the training, with epochs: larger at the beginning and more precise at the end.

- **"Step decay":** Reduce the learning rate by some factor every few epochs.Typical values might be to reduce the learning rate by half every 5 epochs, or by 0.1 every 20 epochs.

In practice: (observe the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) every time the validation error stops improving.

- **"Exponential decay":**

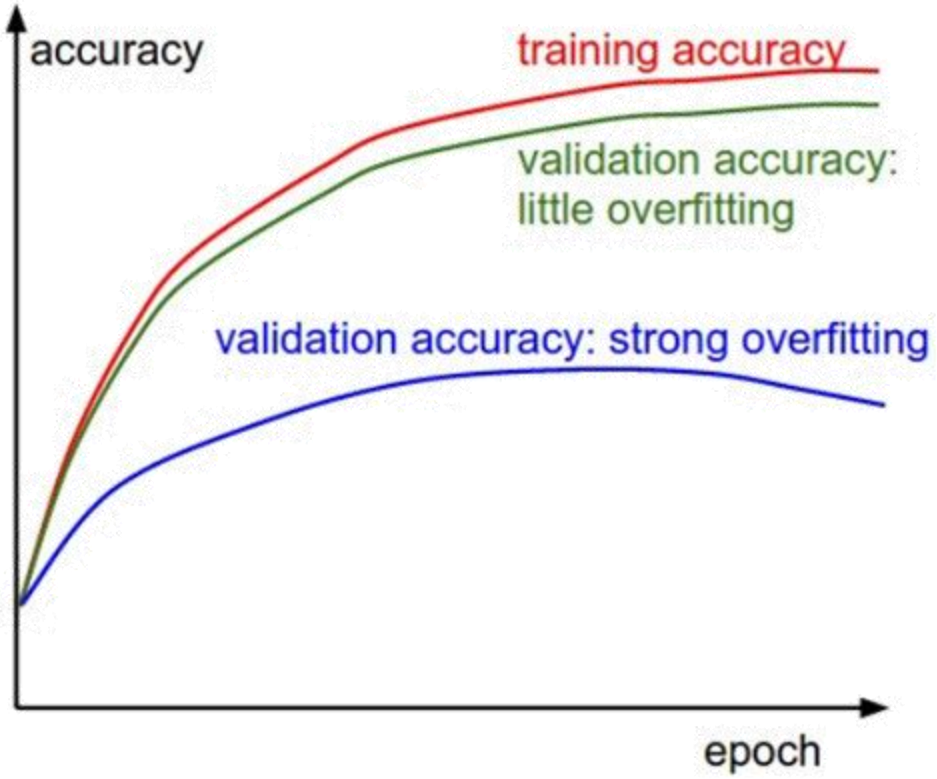
$$\eta = \eta_0 e^{-kt}$$

- **"1/t decay":**

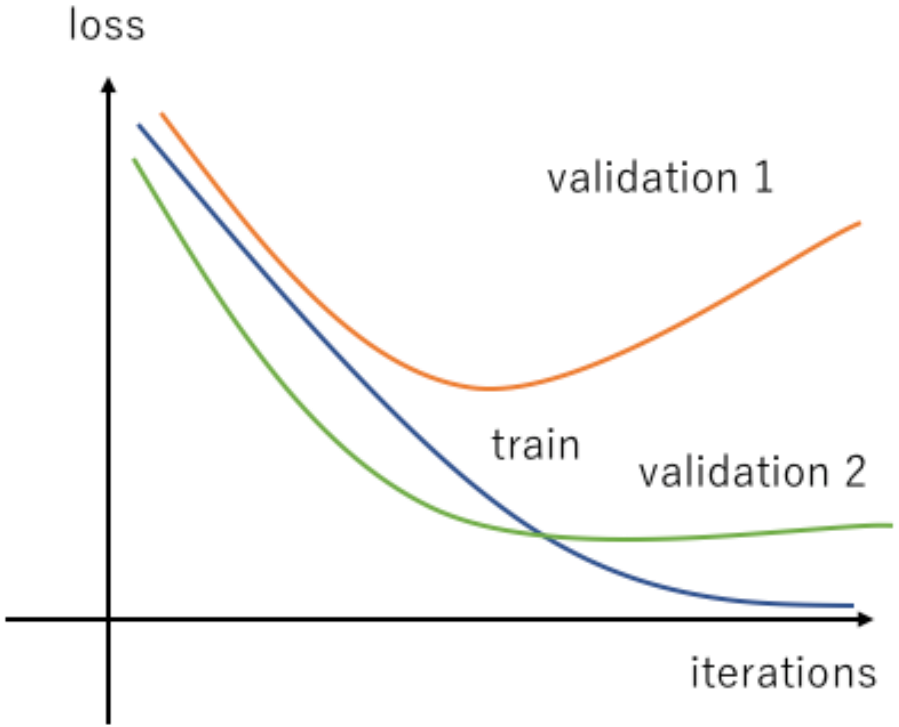
$$\eta = \frac{\eta_0}{1 + kt} e^{-kt}$$

η_0, k: hyperparameters t: number of epoch

Detecting overfitting



Detecting overfitting



Regularization penalty

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$ **Loss without regularization**

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W)$$ **Loss with regularization**

λ :regularization strength

- $\boldsymbol{L_2}$ $\quad R(W) = \sum_i \sum_j W_{i,j}^2$
- $\boldsymbol{L_1}$ $\quad R(W) = \sum_i \sum_j |W_{i,j}|$
- **Elastic Net** $\quad R(W) = \sum_i \sum_j W_{i,j}^2 + \beta |W_{i,j}|$

Update without regularization

$$\mathrm{W}(t+1) = W(t) - \eta \nabla J$$

Update with regularization

$$\mathrm{W}(t+1) = W(t) - \eta \nabla J + \lambda R(W)$$

Drop-oupt

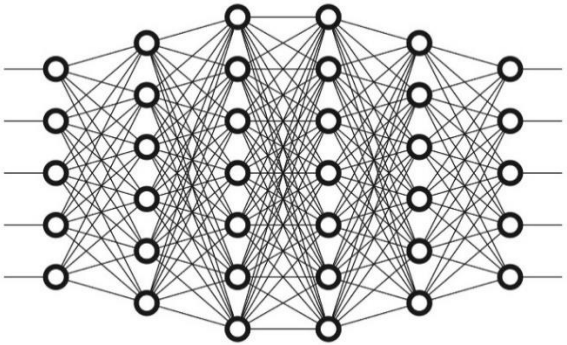
- **“drop-out”:** randomly disconnect inputs from one layer to the next with a probability p: prob. to maintain the link or to disconnect it (implementation dependent)

Without drop-out Drop-out with $p = 0.5$

- For a batch, the links are deactivated for the forward and backward propagation stages, then reactivated and randomly deactivated again for the next batch.
- **dropout only** is used in **training**.
- Disconnecting connections randomly ensures that no single node is always responsible for "activating" when a given pattern occurs. It ensures that there are multiple, redundant nodes that will be triggered when similar inputs are presented, which in turn helps our model to generalise.

KERAS

- Keras is a high-level framework for training neural networks. This library was developed by François Chollet in 2015 with the aim of simplifying the programming of algorithms based on deep learning.
- It offers a more intuitive and high-level set of abstractions. Training can still be done on GPU, remembering that this is the only way we have to train a neural network in an allowable time interval.



Basic layers in Keras


```
keras.layers.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros')
keras.layers.Activation(activation)
keras.layers.Dropout(rate, noise_shape=None, seed=None)
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid')
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',
data_format=None)
keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001,
center=True, scale=True)
keras.layers.Flatten(data_format=None)
```

Keras documentation

https://keras.io/layers/core/

Architecture definition: Sequential mode

Sequential mode (or API): An object of type **Model ()** is instantiated and the layers that make up the architecture are added one after the other.

```
# Imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
# Sequential model
model = Sequential()
# Architecture definition
model.add(Dense(64, input_dim=784))
model.add(Activation('relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

https://keras.io/models/sequential/

Architecture definition: Functional mode

Functional mode (or API): An input is defined and the architecture is defined from these inputs (indicating which is the input to each layer). Once the architecture has been defined, the model object is created by passing it the inputs and outputs (last layer defined).

```
# Imports
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# Input definition
inputs = Input(shape=(784,))
# Architecture definition
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
# We create the model object as a union of inputs and architecture
model = Model(inputs=inputs, outputs=predictions)
```

https://keras.io/models/model/

Compiling a Keras model

Before training the model in Keras, it is necessary to compile it by configuring the training process. This process is carried out by means of the command **model.compile**.

```
# Example for a multi-class classification problem
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Example for a binary classification problem
model.compile(optimizer=SGD(lr=0.01, decay=1e-6, momentum=0.9,
nesterov=True),
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Example for a regression problem
model.compile(optimizer='rmsprop',
              loss='mse')
```

Training in Keras

Once the model has been compiled in Keras it is possible to launch the training process. Keras trains models from data and labels stored in Numpy arrays using the **model.fit** method.

```
# Data generation
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))
# Example with labels in one-hot encoding
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',
              metrics=['accuracy'])
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
# Example with categorical labels
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels, epochs=10, validation_data=(data_v, labels_v),
batch_size=32)
```

Prediction and evaluation in Keras

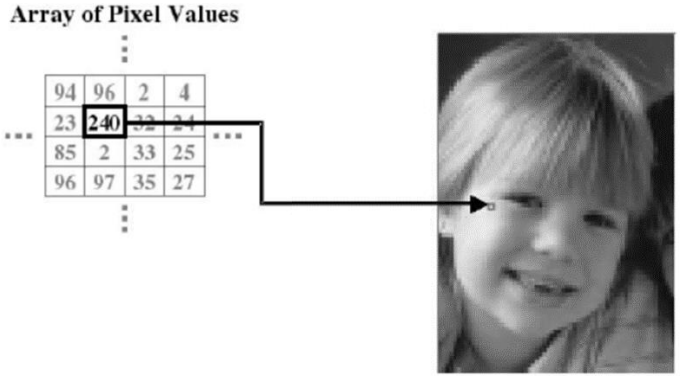
Once the trained model is available, an evaluation of the model must be carried out. To do this, the test data is predicted with the **model.predict** command and then performance metrics are obtained or **model.evaluate** is used

```
# Example with model.evaluate
model.evaluate(x=X_test, y=y_test, batch_size=None)
%% Returns the values of losses and metrics used in training for the test data.
# Example with model.predict and evaluation_report
from sklearn.metrics import classification_report
predictions = model.predict(x=X_test, batch_size=None)
print(classification_report(y_labels,predictions.argmax(axis=1))
%% Returns more metrics
```

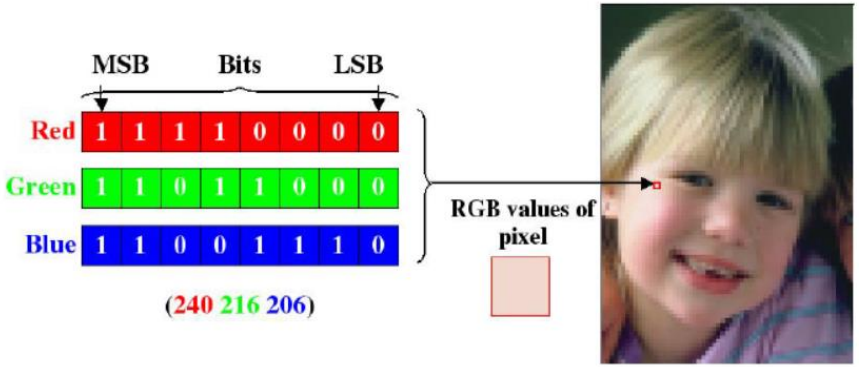
Contents

Concept of image

Gray image



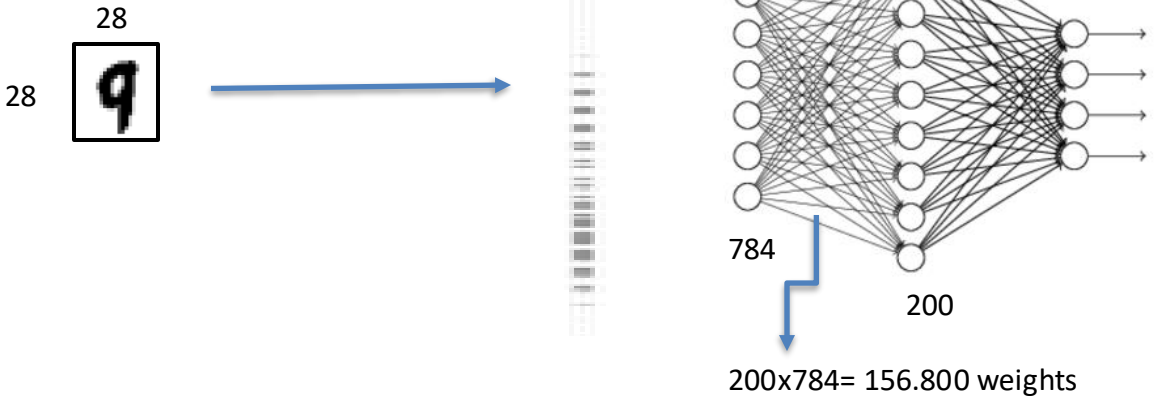
RGB image



level: 0….255 (black….white)

Problem with images

Example: In the MNIST database for image classification



- When the size of the image starts to grow, using fully connected layers (one neuron per pixel) becomes unfeasible.

Hand-crafted learning

Hand-crafted learning

Data

Feature Extraction

Granulometry

SIFT

Colour

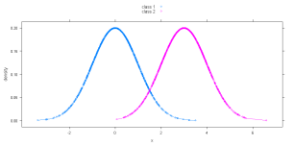
LBP

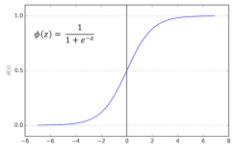
SURF

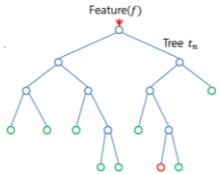
HOG

Co-occurrence

Classification or Regression



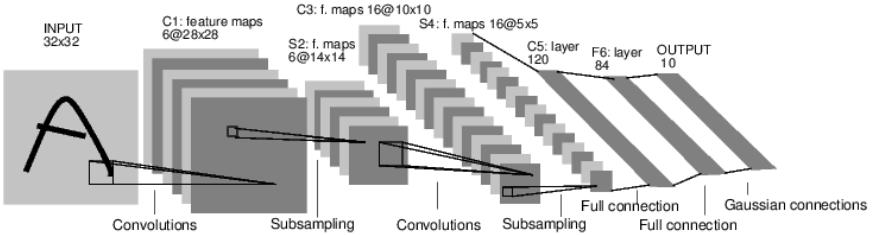




Automated learning

(Data)^2

Deep neural networks

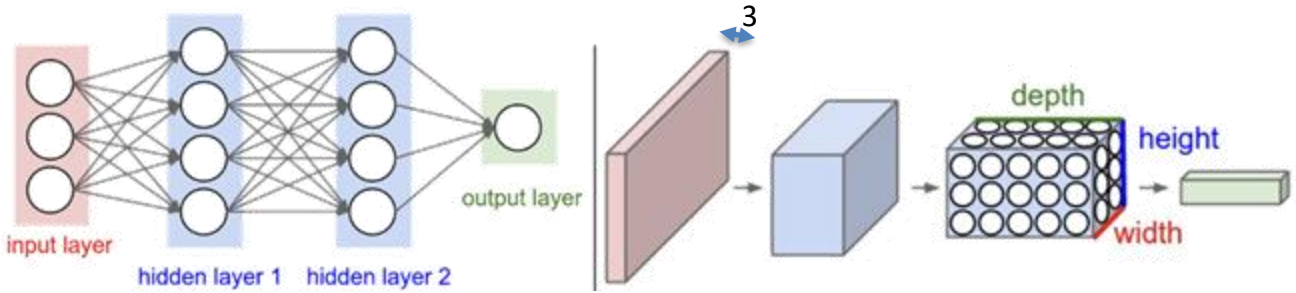


Feature extraction

Classification

Convolutional neural networks

- Convolutional networks are a specialized kind of neural network for processing data that has a grid-like topology. They take advantadge of:
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units



Multilayer Perceptron

Convolutional neural network

Convolution

$$y[m,n] = x[m,n] * h[m,n] = \sum_k \sum_l x[k,l]h[m-k,n-l]$$

74	66	62	59	63	71	75	72
72	65	61	58	62	70	75	72
71	68	63	59	63	69	73	69
70	67	62	59	62	69	73	69
70	64	59	57	61	70	74	72
68	66	64	63	65	68	69	68
70	69	67	66	67	69	69	67
69	68	66	64	65	67	66	63

Imagen de salida

			3				

Kernel

$h[k,l]$

1	0	-1
2	0	-2
1	0	-1

$h[-k,-l]$

-1	0	1
-2	0	2
-1	0	1

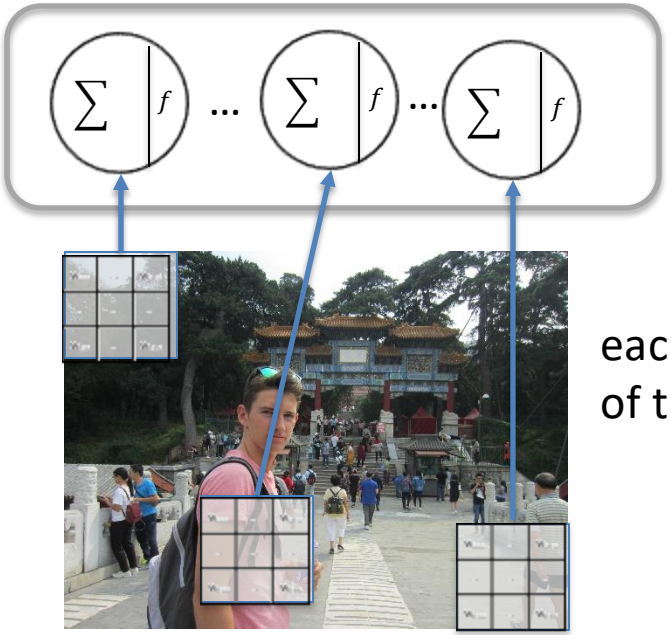
$$(-1)*62 + 0*59 + 1*62 + (-2)*59 + 0*57 + 2*61 + (-1)*64 + 0*63 + 1*65 = 3$$

Convolution

1	0	-1
2	0	-2
1	0	-1

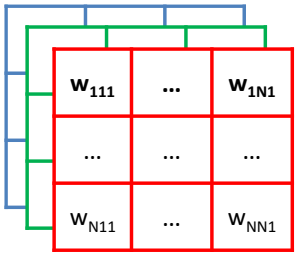
1	2	1
0	0	0
-1	-2	-1

Local connectivity



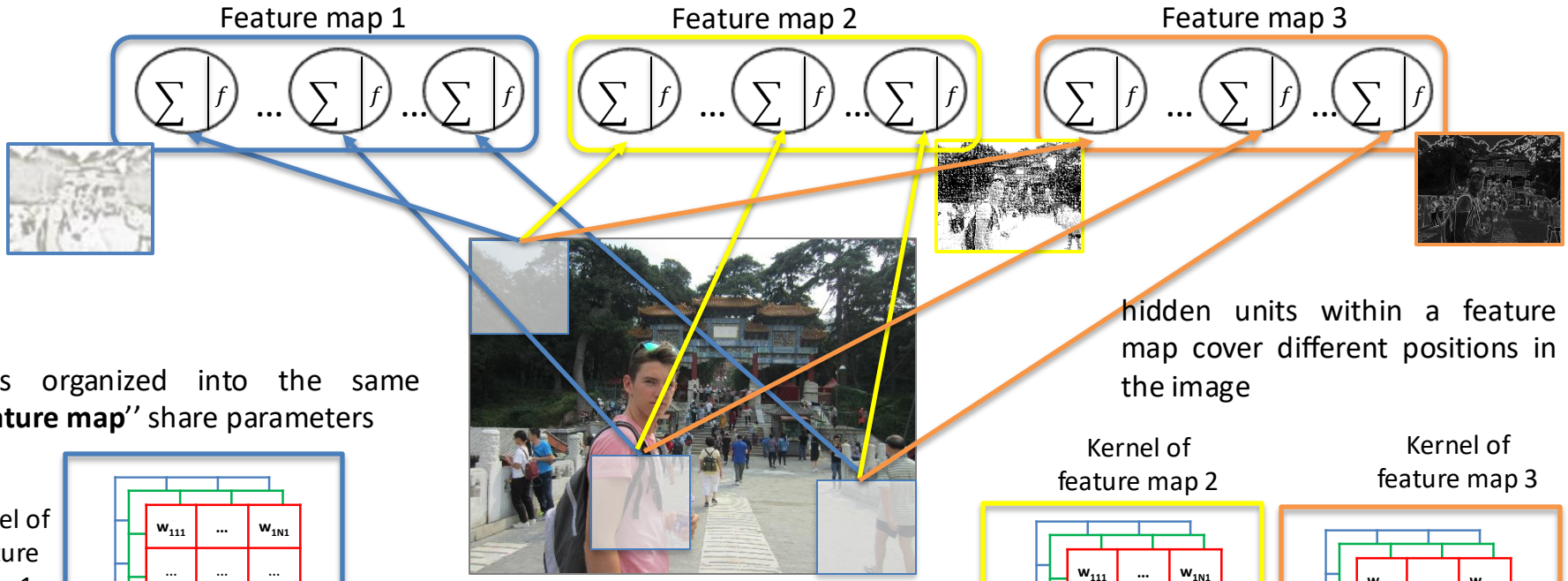
each hidden unit is connected only to a subregion (patch) of the input image: receptive field

w_{111}	...	w_{1N1}
...	...	...
w_{N11}	...	w_{NN1}



Receptive field

Parameter sharing

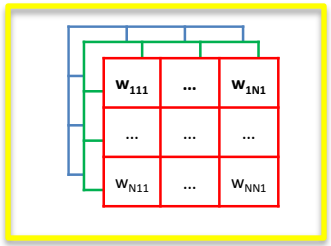


units organized into the same ‘‘**feature map**’’ share parameters

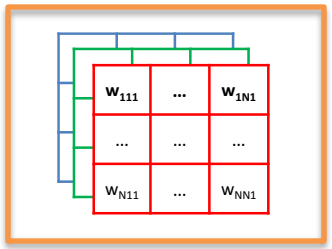
Kernel of feature map 1

hidden units within a feature map cover different positions in the image

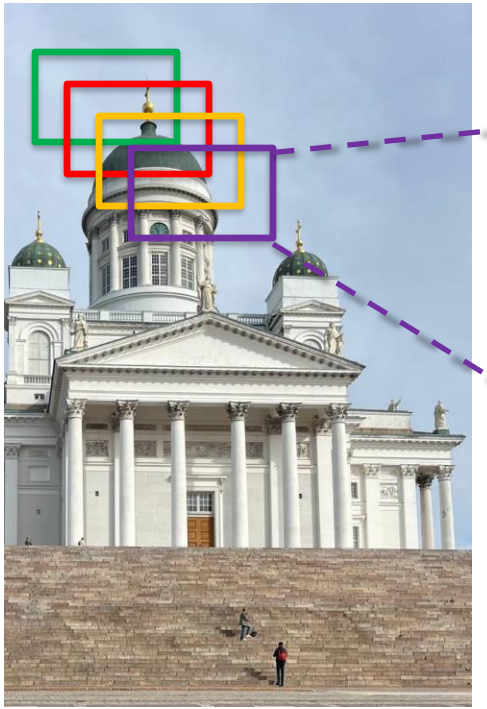
Kernel of feature map 2



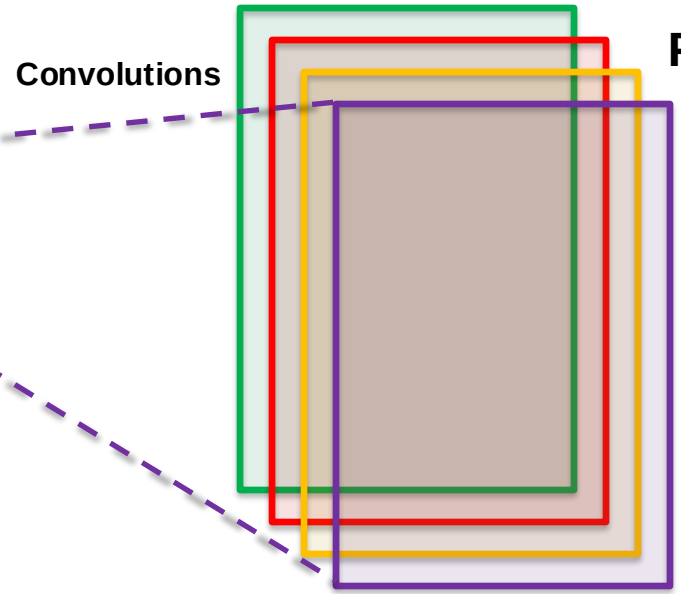
Kernel of feature map 3

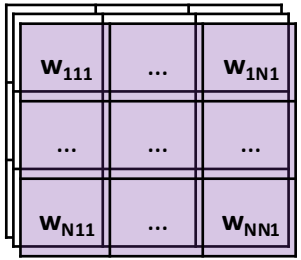


Feature maps



Input image

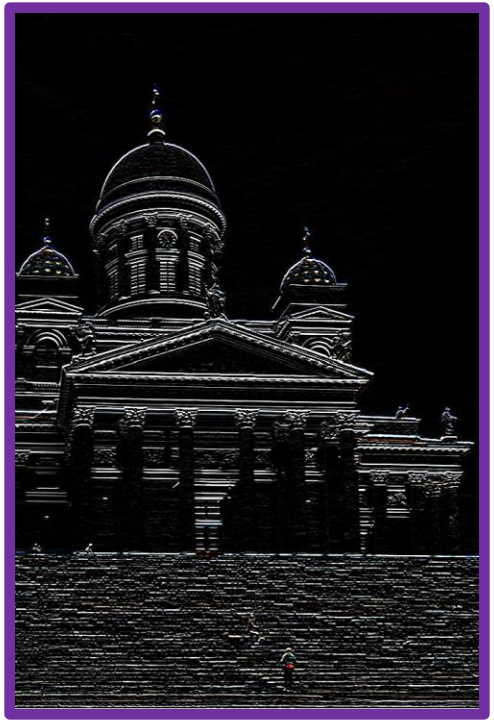




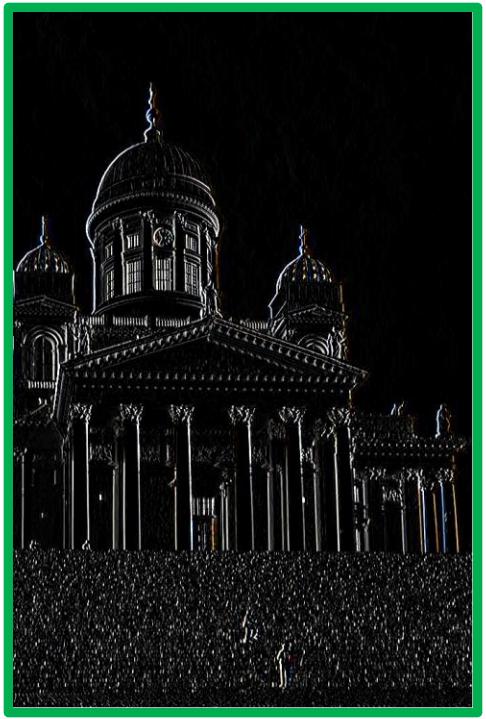
Learnable

Feature maps

Input image

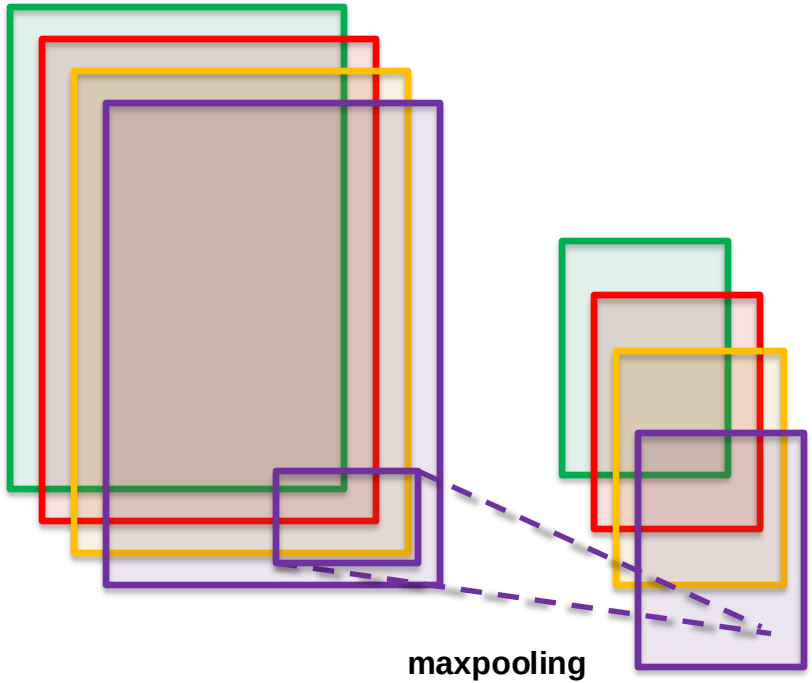


Feature map 1



Feature map 2

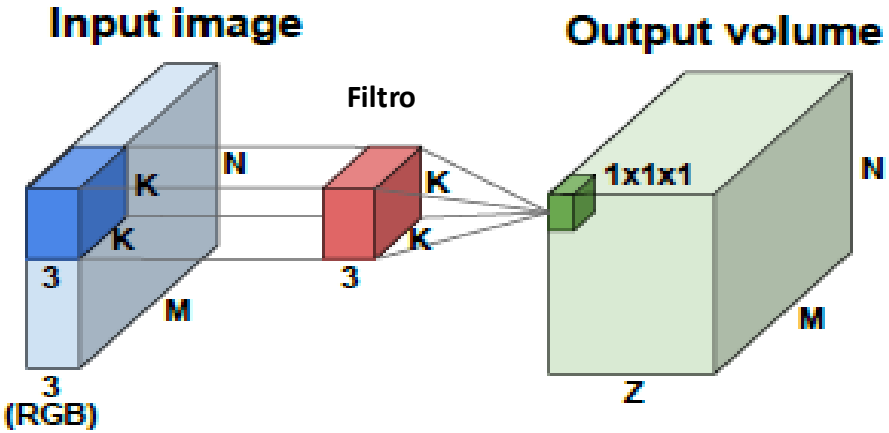
Pooling



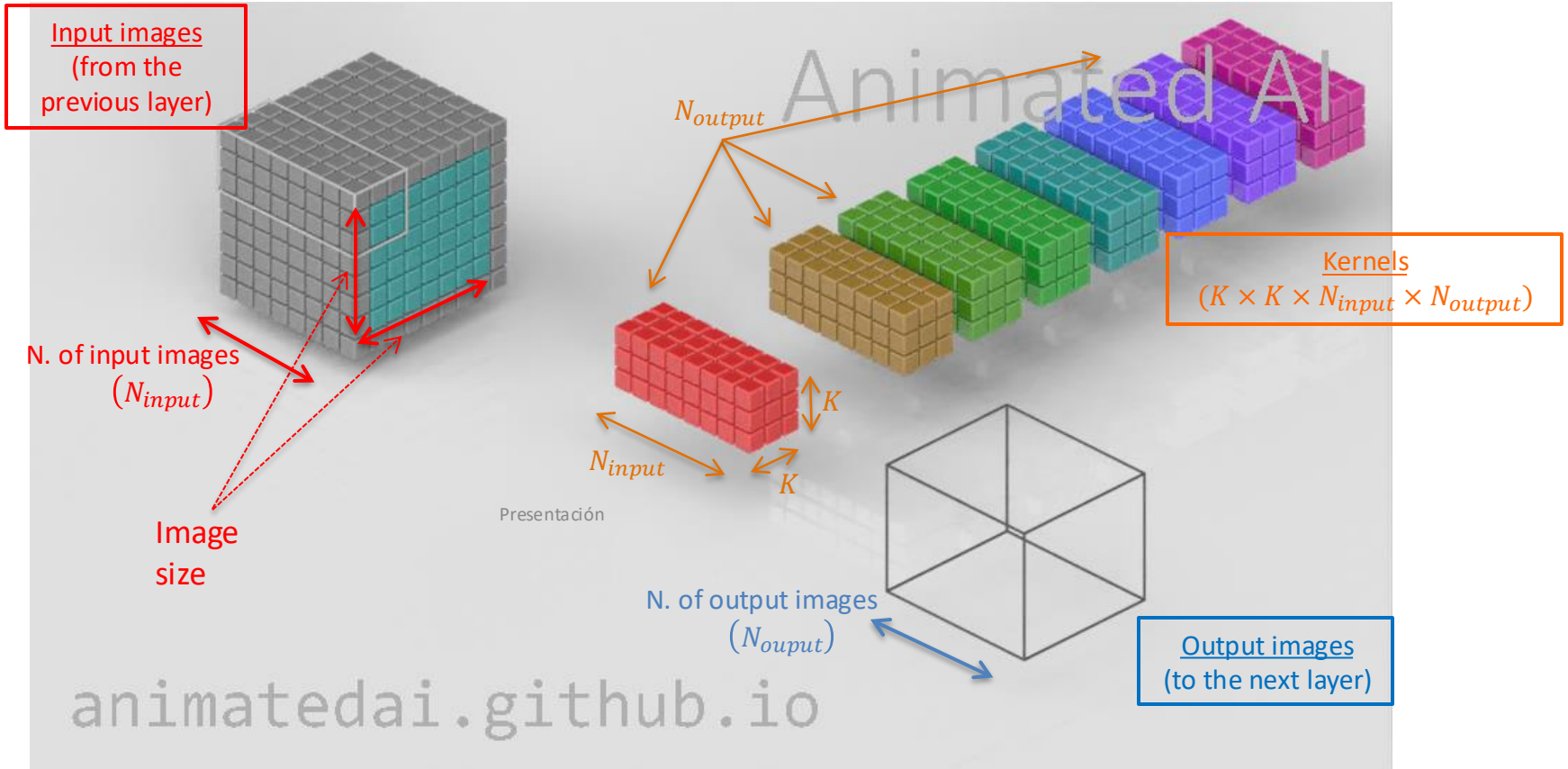
Pooling is performed in non overlapping neighborhoods (subsampling)

Contents

Convolutional layers

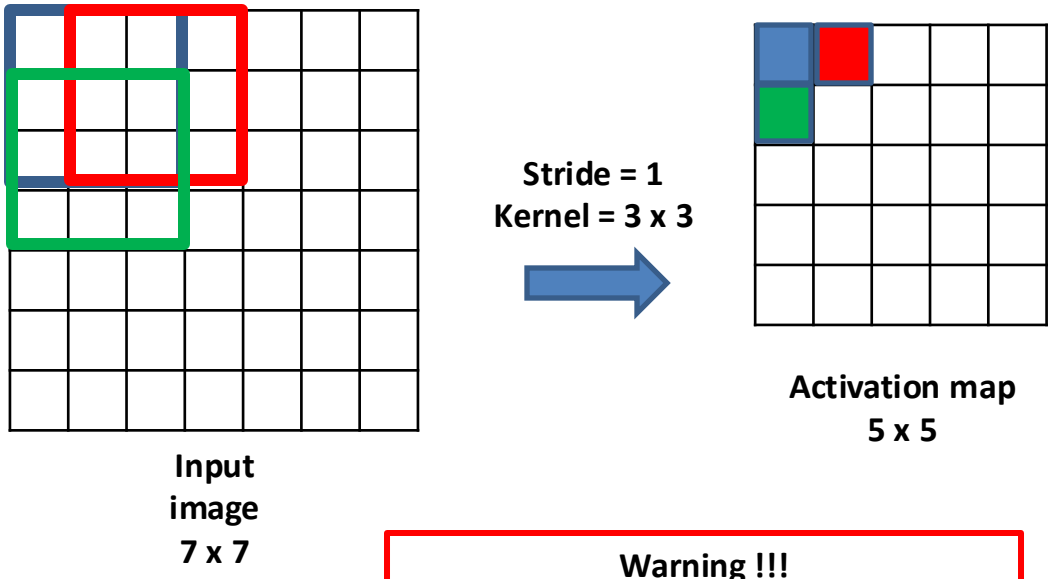


Convolutional layers



Convolutional layers: Stride

- The **center of the kernel** (or mask filter) moves to the next pixel in steps given by the **Stride** hyperparameter

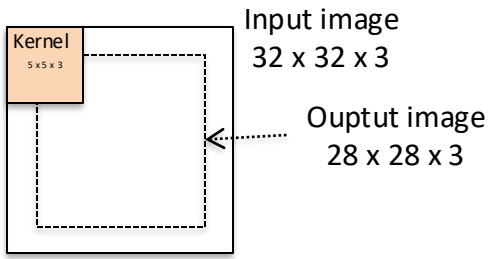


Warning !!!
Reduction of the image size.
It depends on kernel size and stride

Convolutional layers: zero padding

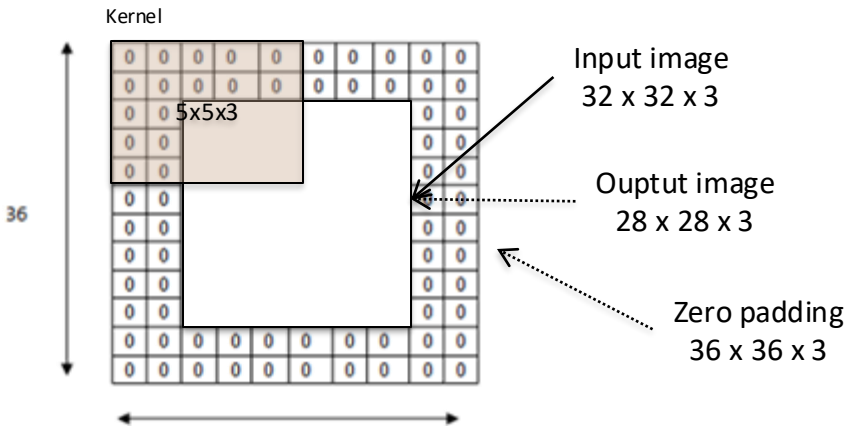
What happens when a 5 x 5 x 3 filter is applied to a 32 x 32 x 3 input volume (32 x 32 pixel RGB image)?

Size of activation map: 28 x 28 x 3



In the first layers of the network we want to preserve as much information as possible from the original image.

Zero Padding

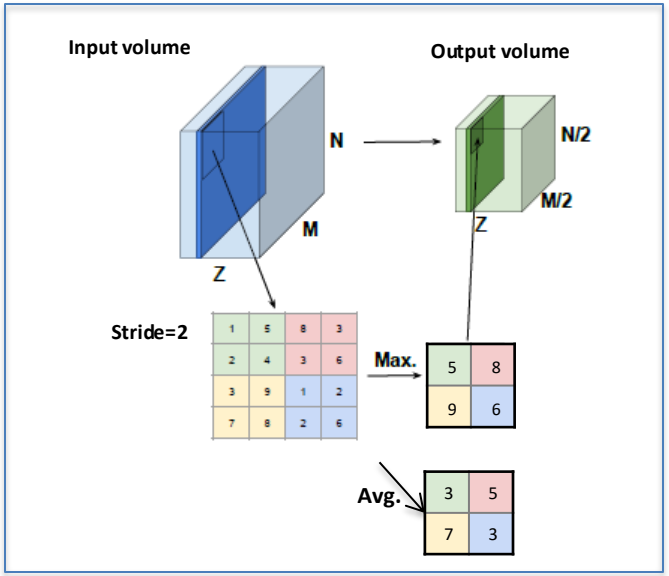


Pooling layers

- This layer is a decimation layer in spatial dimension (width, height)
- Its function reduces the spatial size of the representation :
  - Reduce the number of paremeters
  - Reduce computational cost
  - Avoid overfitting

Typical hyperparameters:
- Filter size = 2×2
- Stride = 2

Main functions:
- Max pooling
- Average pooling
- L2-norm pooling



CNN. Setup

- Among the different layers, some aditional parameters have to been considered :
  - Convolutional layer
    - Kernel size
    - Number of filters
    - Stride
    - Zero padding
  - Pooling layer
    - Stride
    - Window size
  - Activation layer
  - Batch normalization layer
  - Drop out layer
  - Classification: Top model

Basic layers in Keras


```
keras.layers.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros')
keras.layers.Activation(activation)
keras.layers.Dropout(rate, noise_shape=None, seed=None)
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid')
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',
data_format=None)
keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001,
center=True, scale=True)
keras.layers.Flatten(data_format=None)
```

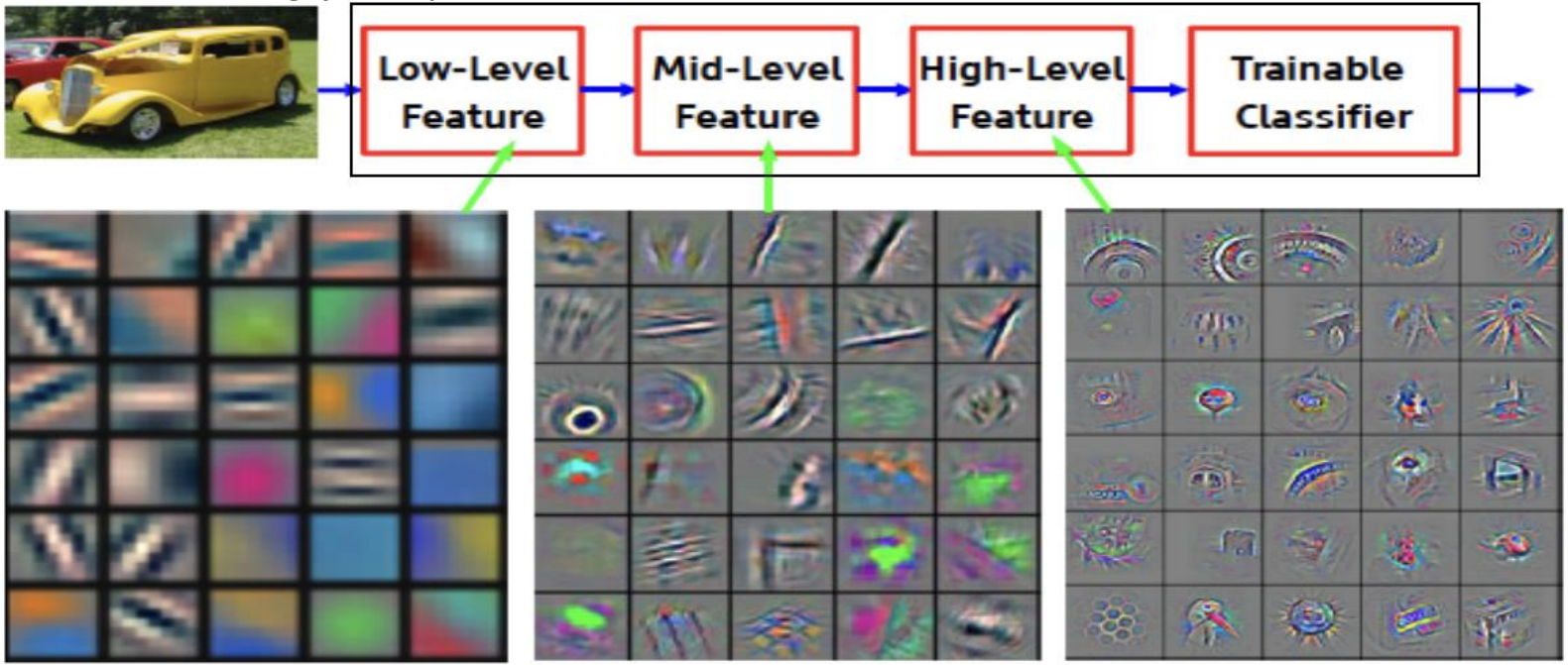
Keras documentation

https://keras.io/layers/core/

Contents

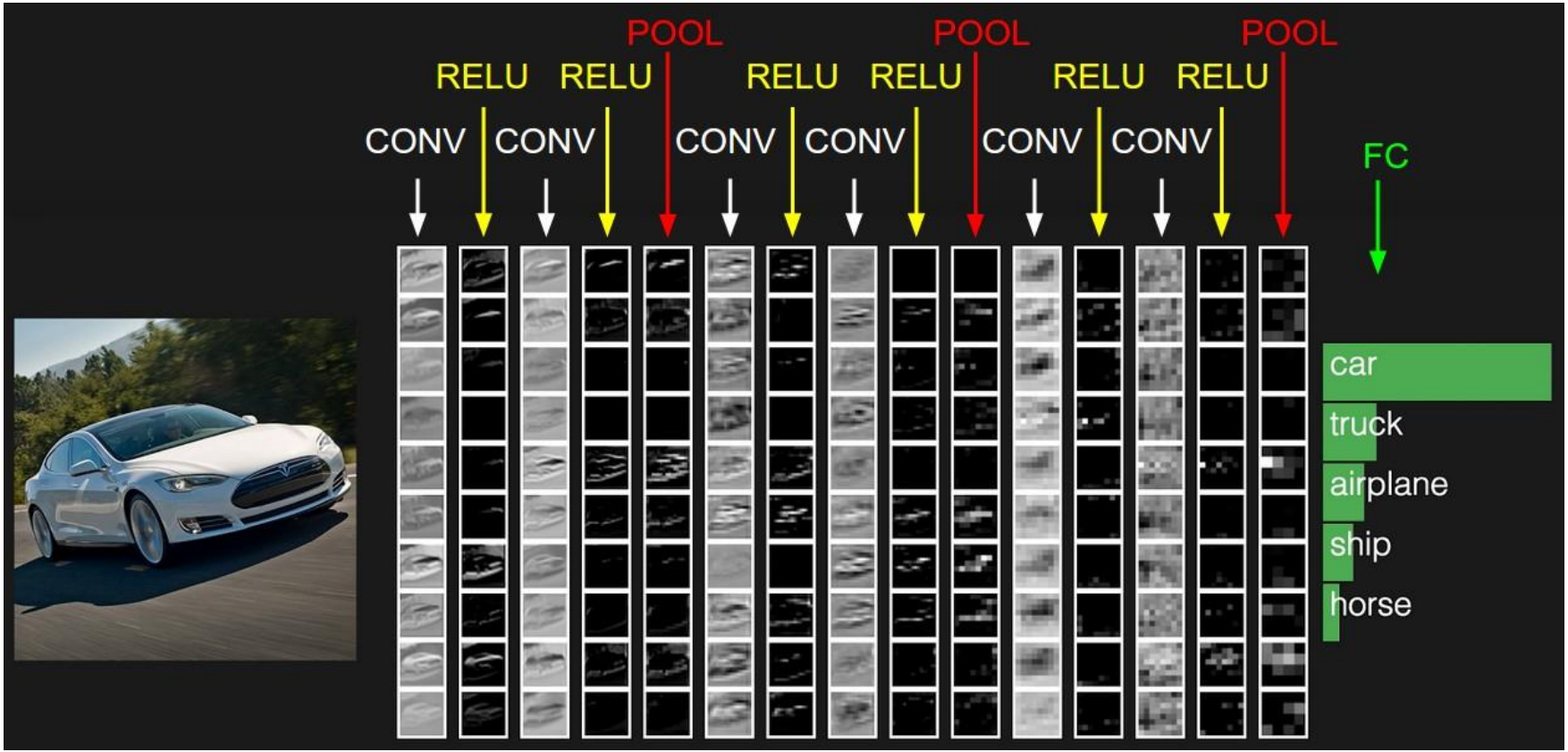
CNN: Feature extraction

- CNN = learning hierarchical representations with increasing levels of abstraction
- End to end training: joint optimization of features and classifier



Feature visualization of convolutional net trained on ImageNet (Zeiler, Fergus, 2013)

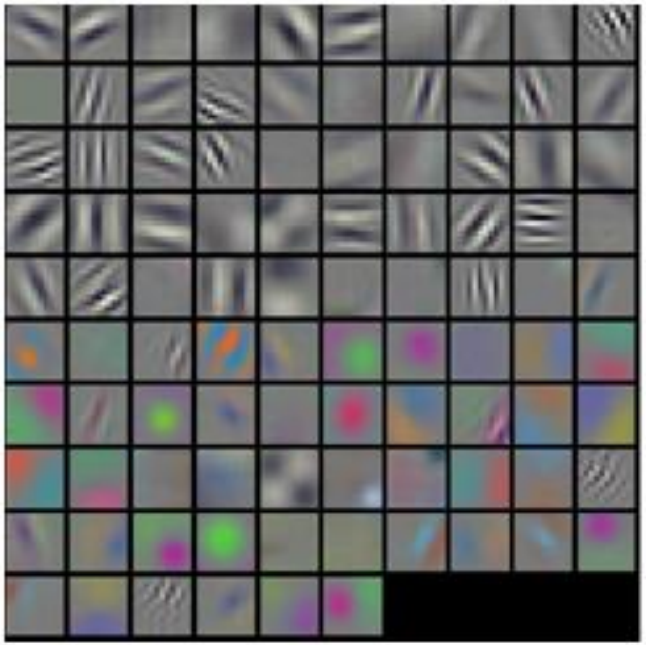
Feature extraction



http://cs231n.stanford.edu/

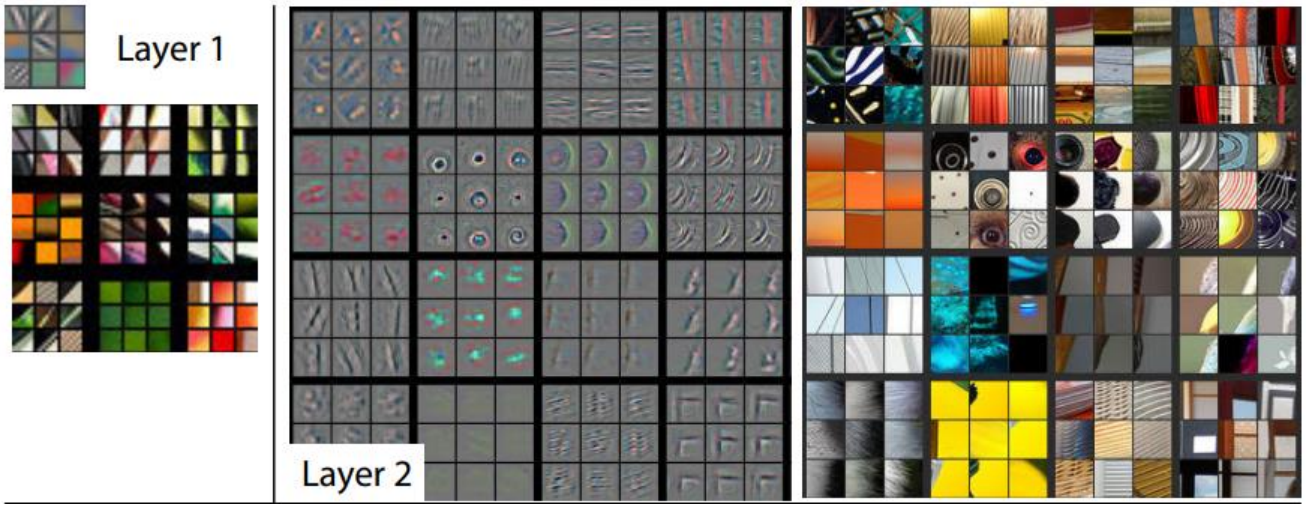
Low-level features

- How does the network detect low-level features?

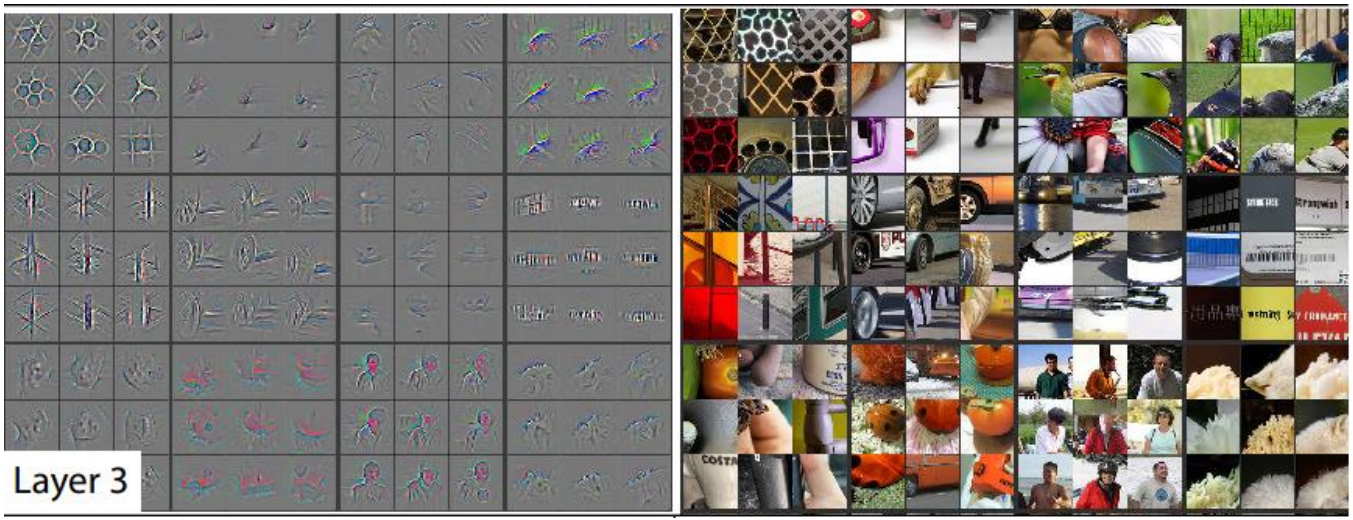


- Convolutional blocks: The output (activation map) of the first convolutional layer will be the input of the second layer, and so forth.
- The output of the second convolutional layer will be the activations representing higher level features: semicircles (combination of a curve and a straight edge), squares (combination of several straight edges), etc.

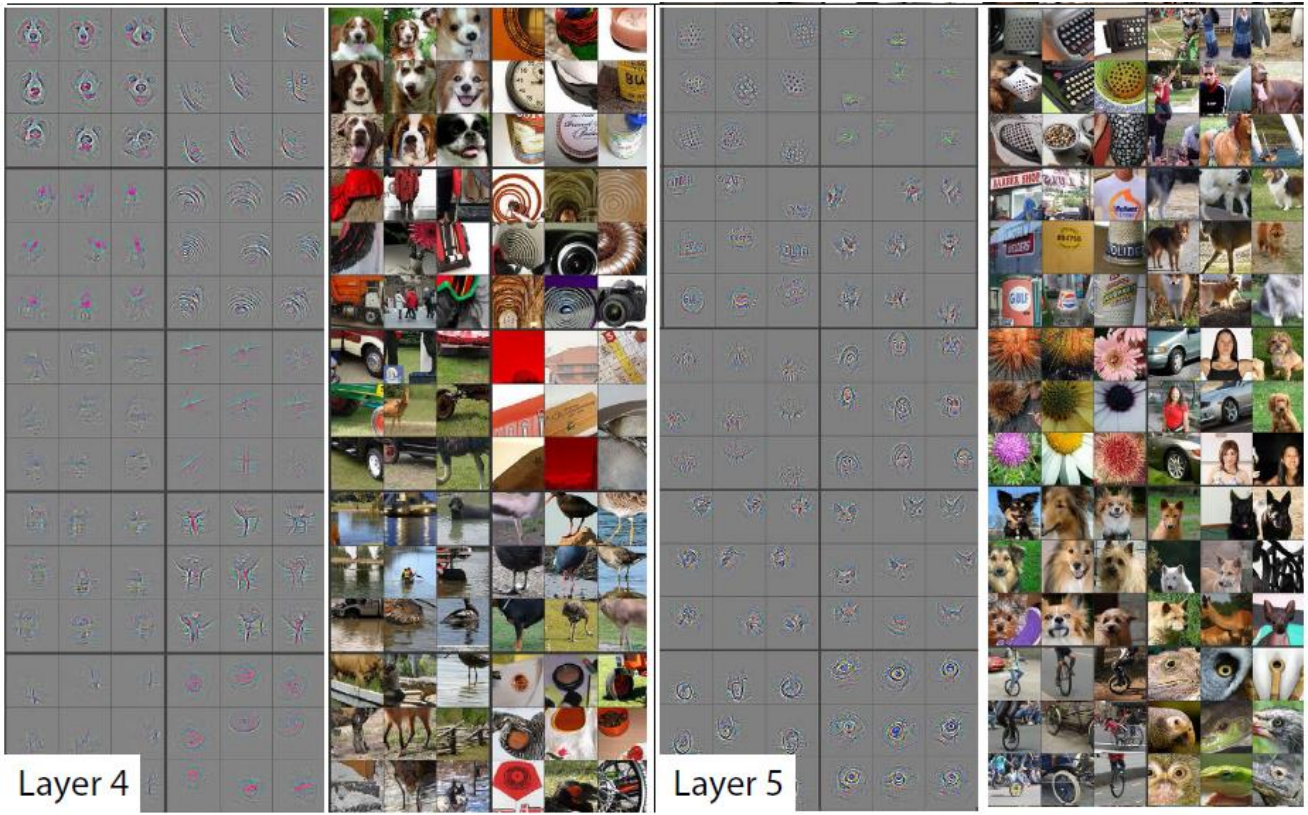
High-level features



High-level features



High-level features

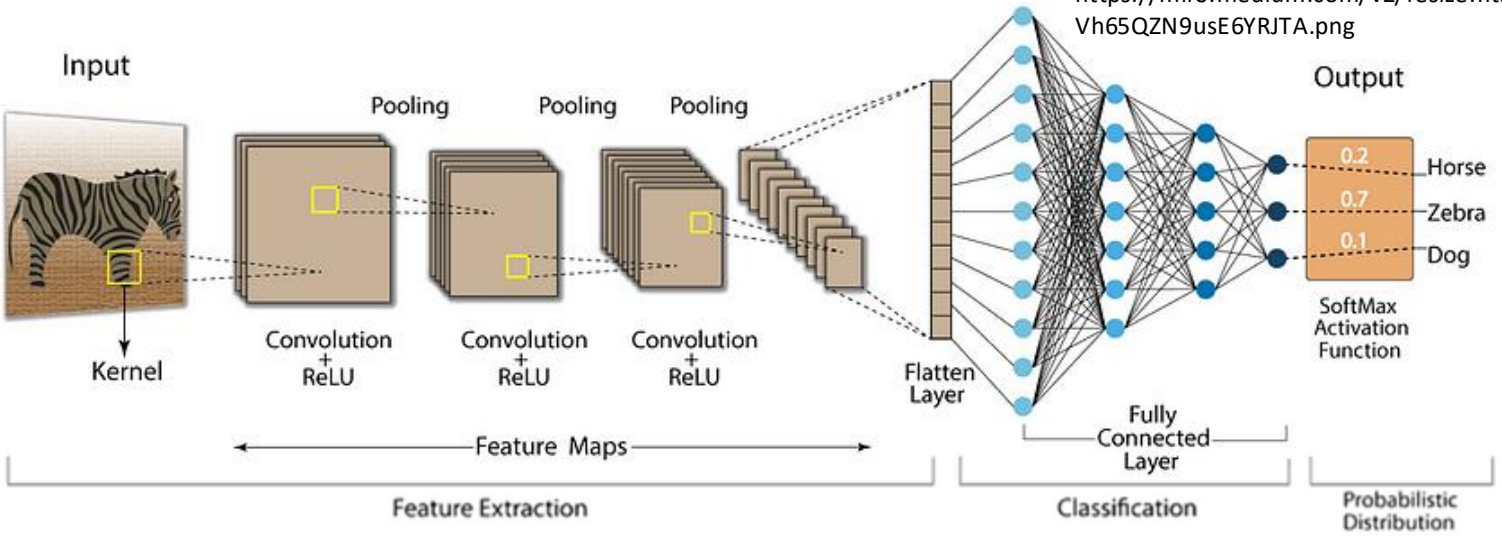


https://poloclub.github.io/cnn-explainer/

Contents

CNN for classification

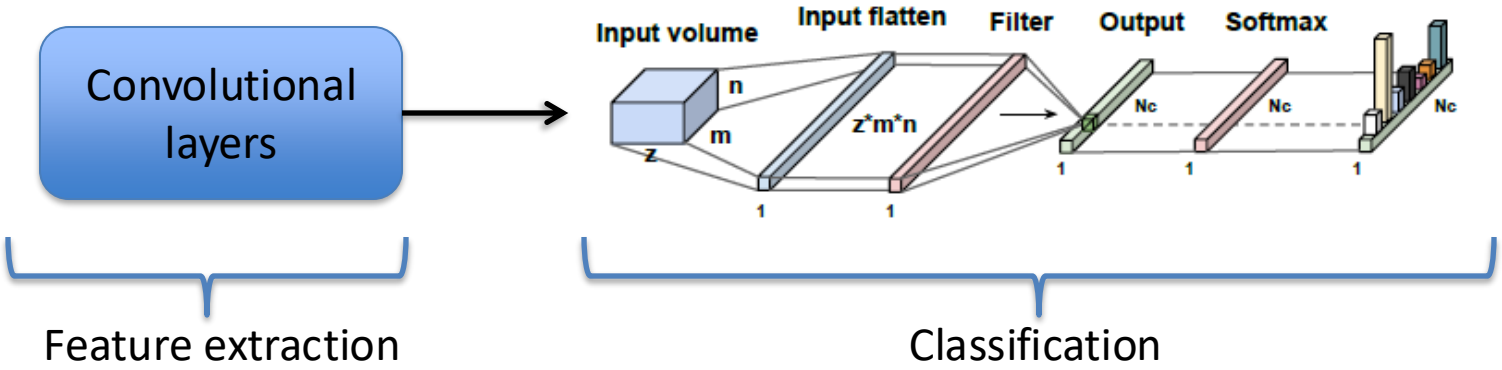
https://miro.medium.com/v2/resize:fit:847/1*eiWpSVh65QZN9usE6YRJTA.png



- Training: stochastic gradient descent/ forward-backward propagation
- Hyperparameters setting: as MLP

CNN for classification

- Top model:
  - Flatten layer
  - Multilayer perceptrón
  - The last fully connected layer is a softmax layer (in the case of more than one output neuron) with as many neurons as the number of classes.



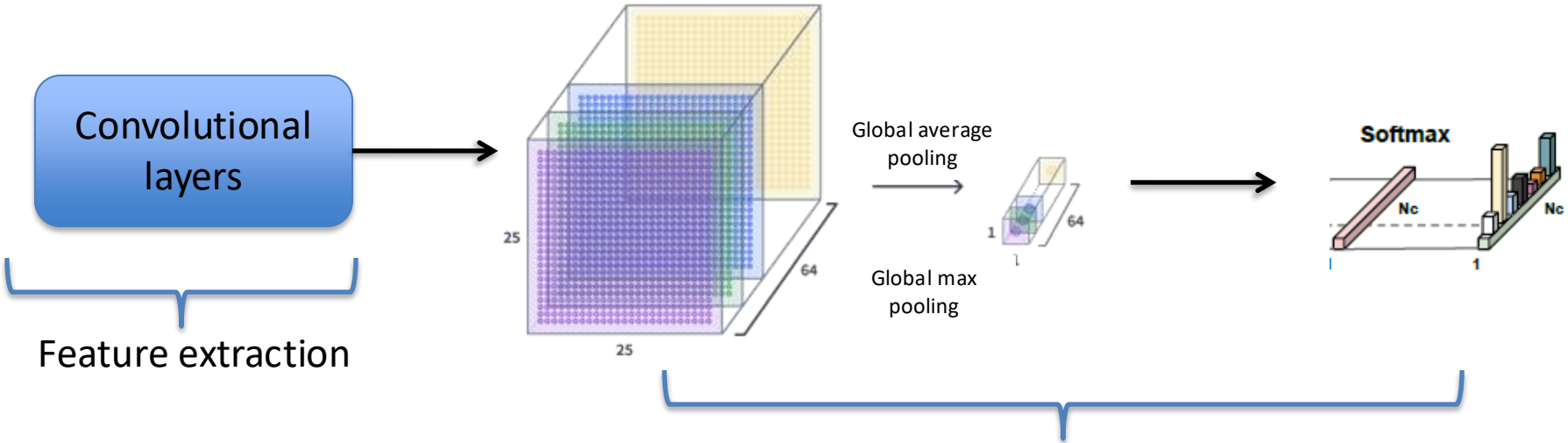
- PROBLEM: Overfitting

CNN for classification

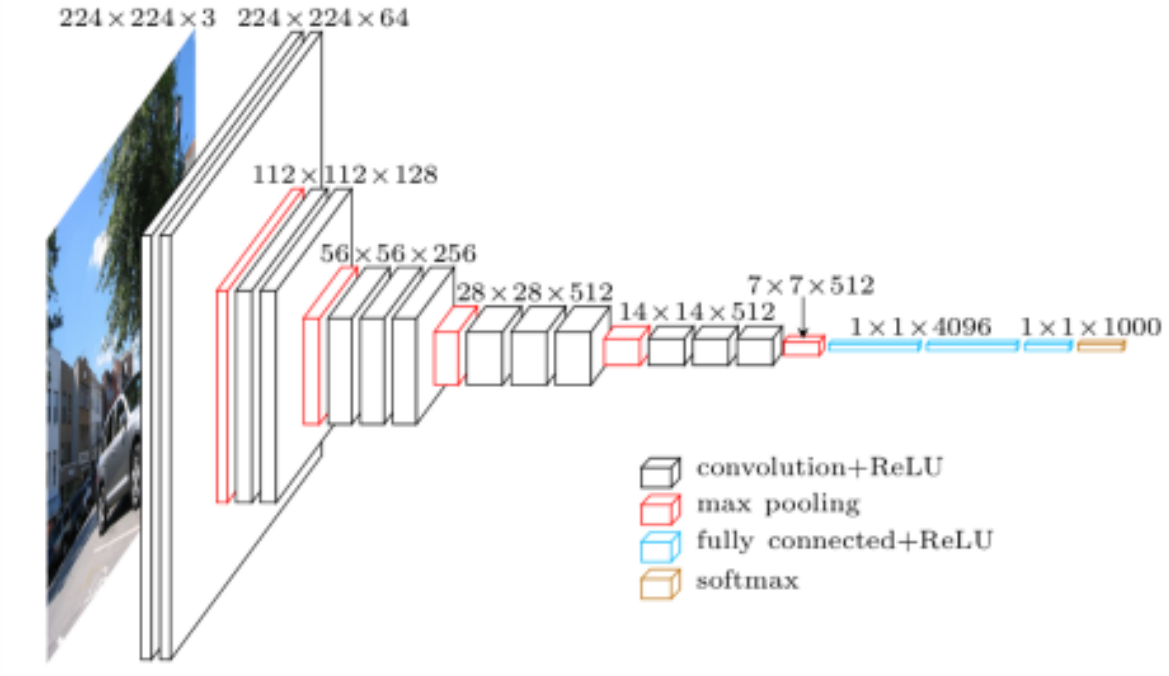
- Top model:
  - Global averagepooling + softmax
  - Global maxpooling+ softmax

GAP: calculates the average output of each feature map

GMP: calculates the maximum output of each feature map



CNN. Final Global Architecture

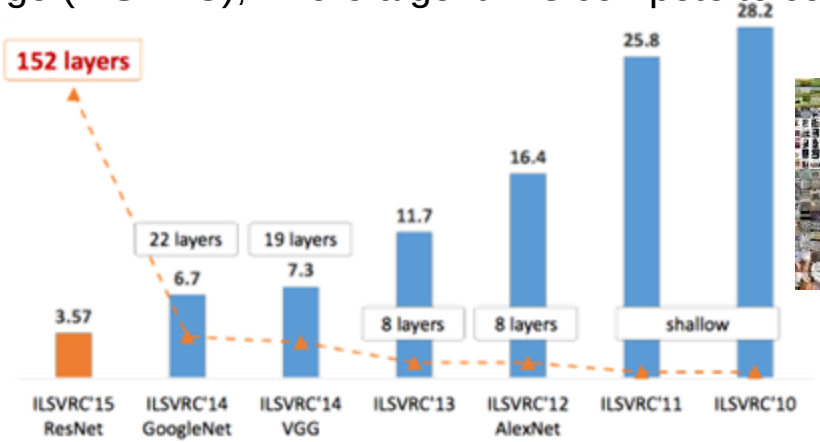


Contents

ImageNet

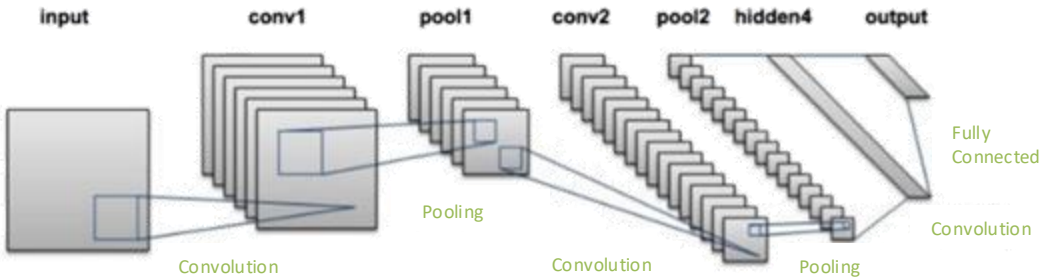
- **ImageNet** is a database containing **14 million images** classified into **1000 different categories** (http://www.image-net.org/).
- The most popular state-of-the-art architectures have been trained with ImageNet and the resulting network weights are **publicly available.**
- The **ImageNet Project** runs an annual competition, the ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**), where algorithms compete to correctly classify objects and scenes.

Objetive: to reduce training parameters while maintaining or exceeding the previous year results, with a strong focus on reducing computational cost.



LeNet-5

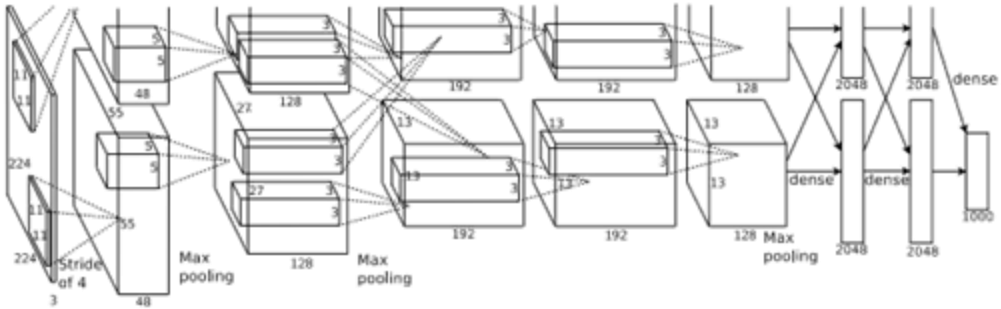
- LeNet-5 (1998, Lecun et al.): **CNN pioneer**.
- **7 layers**
- Aplication: digit classification (banks to recognise handwritten digits on cheques).
- Graylevel images of **28x28 pixels**. Higher resolution images: more convolutional layers.



ALexNet

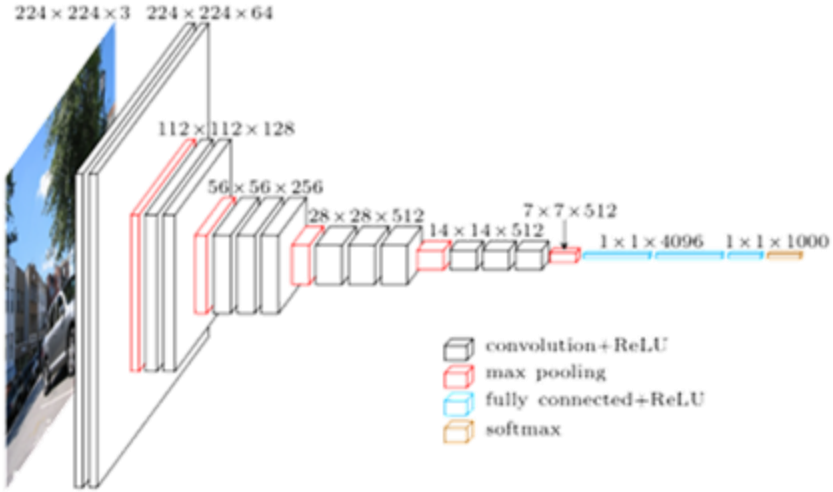
- Winner of the **2012 ImageNet challenge** and significantly outperformed all competitors by reducing **the top-5 error** from 26% (runner-up) to **15.3%**. SuperVision group.

- Deeper than LeNet and with more filters per layer.
  - 11x11, 5x5,3x3 convolutional filters and RELU activations.
  - Max pooling layers
  - Dropout
  - Data augmentation
  - SGD with momentum
  - Trained for 6 days using 2 Nvidia Geforce GTX 580 GPUs.



VGG16 y VGG19

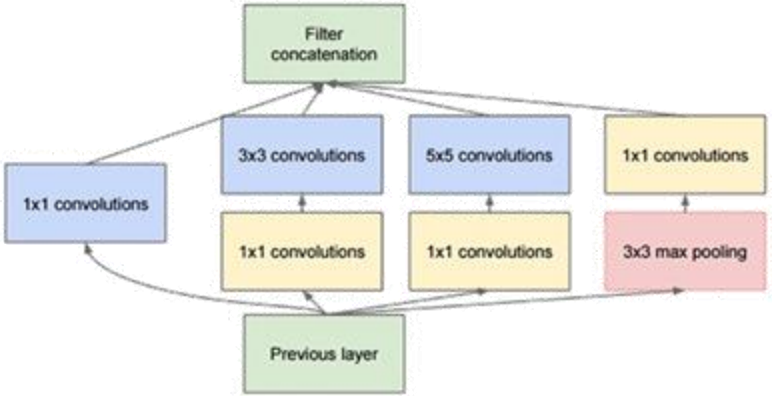
- Fairly simple architecture: **blocks composed of an incremental number of convolutional** layers with **filters of size 3x3** with interleaved **maxpooling layers** (halving the size of the activation maps).
- Classification block: **2 fully-connected layers** (4096 neurons) and the output layer (1000 neurons).
- 16 and 19: differences in number of weighted layers in each network (convolutional and fully connected).



Simonyan y Zisserman (2014). Very Deep Convolutional Networks for Large Scale Image recognition (https://arxiv.org/abs/1409.1556)

Inception V3 (GoogleNet)

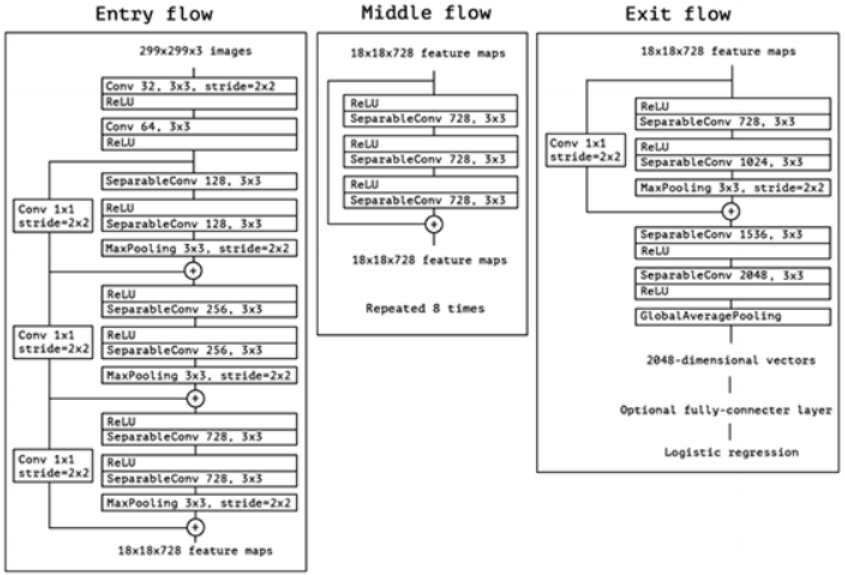
- This type of architecture, introduced in 2014 by Szegedy et al. **"Going Deeper with Convolutions**" (https://arxiv.org/abs/1409.4842), uses blocks with filters of different sizes that are then concatenated in order to extract features at **different scales** that are then combined into a single activation map.



- Challenge winners in **2014, top-5 error rate of 6.67%!**
- This architecture, **22 convolutional layers**, requires less memory than VGG and ResNet. It reduces the number of parameters from 60 million (AlexNet) to 4 million.

Xception

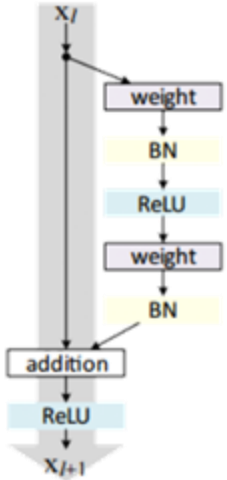
- Proposed **by François Chollet** (creator of Keras). Like Inception but provides a quick way to make **2D convolutions (2 1D convolutions).**



"Xception: Deep Learning with Depthwise Separable Convolutions", https://arxiv.org/abs/1610.02357.

ResNet (Microsoft)

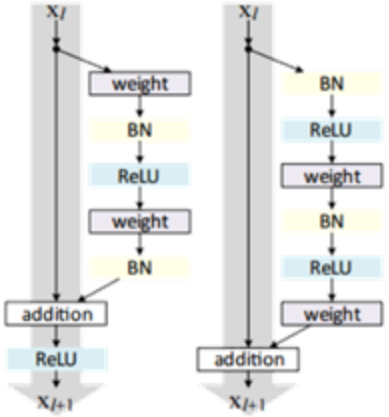
- Developed by He et al. in 2015 (**2015 winners**): introduces an exotic type of architecture based on modules ("**networks within networks**").
- Introduces the concept of "**residual connections**". In layer l+1 the activation map of the unmodified and modified l-layer l+1 is combined..



"Deep Residual Learning for Image Recognition": https://arxiv.org/abs/1512.03385

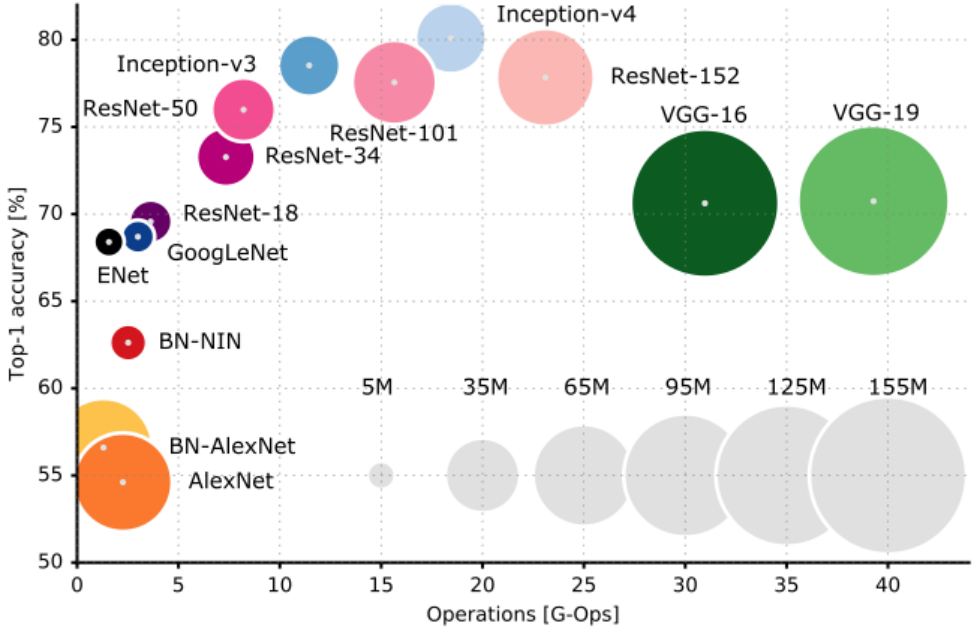
ResNet

- **Improved** architecture **in 2016** improved by including more layers of residual blocks.



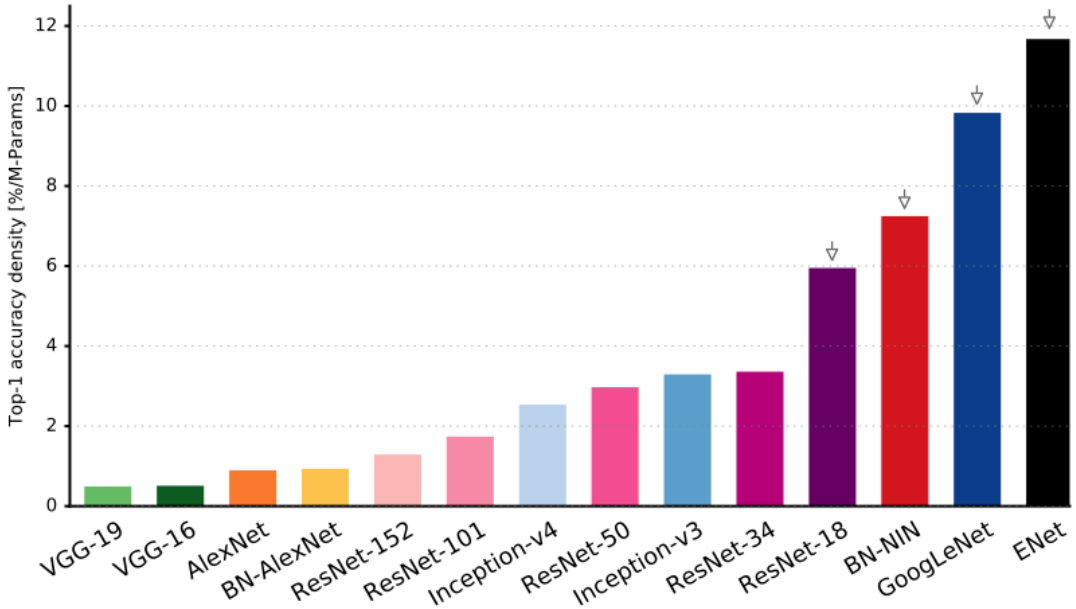
- There are **variations of ResNet** with different numbers of layers, but the most widely used is **ResNet50**, which consists of 50 with weights.
- Many more layers than VGG but needs almost 5 times less memory: fully connected layers are replaced by a type of layer called **GlobalAveragePooling**, which converts 2D activation maps to a vector of n classes that is used to calculate the probability of belonging to each class**.**

Comparison of sizes



Source: ***AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS, 2017***

Comparison: accuracy vs number of parameters

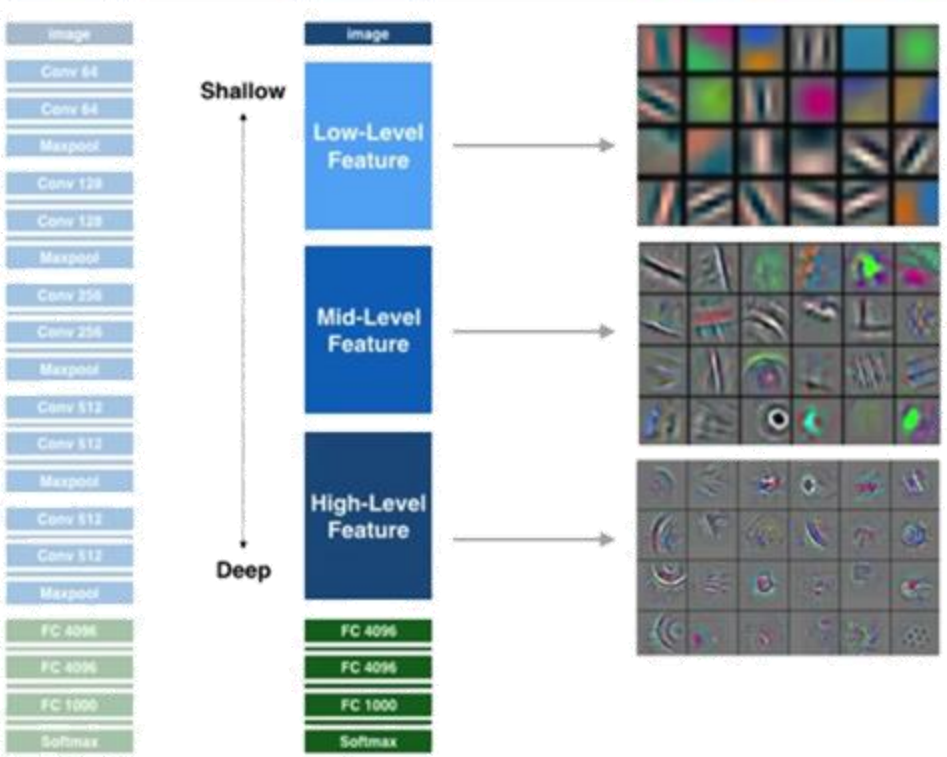


Source: ***AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS, 2017***

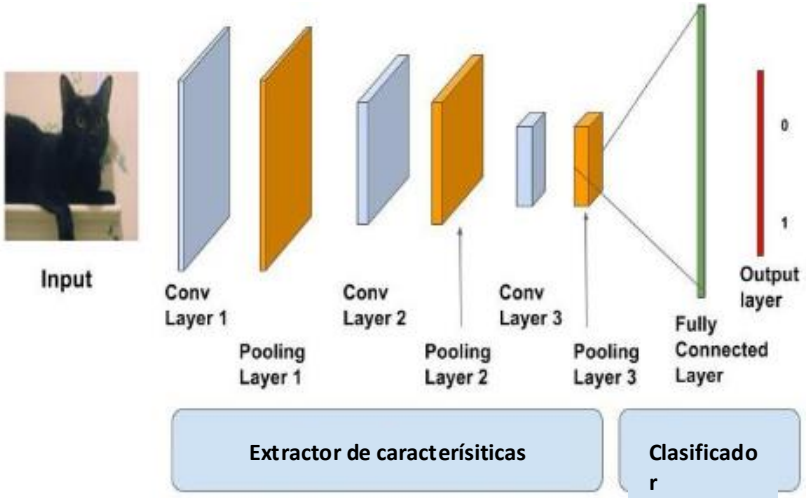
Transfer learning

- **Training** a neural network is **costly** (time).
- **Transfer learning** techniques to avoid:
  - Define the architecture of a neural network.
  - Train it from the beginning.
- **Idea**: Initialise the weights of a predefined network with values that classify a given dataset well and tune them to our problem.
- **We avoid**:
  - **Need** for a **dataset as large** as necessary if we want to train a network from scratch (from hundreds of thousands or even millions of images we could go to a few thousand).
  - **Need to wait** a good number of epochs to get values for the optimal weights for classification.
- Two techniques:
  - **Transfer learning**
  - **Fine-tuning**

Transfer Learning vs Fine-Tuning

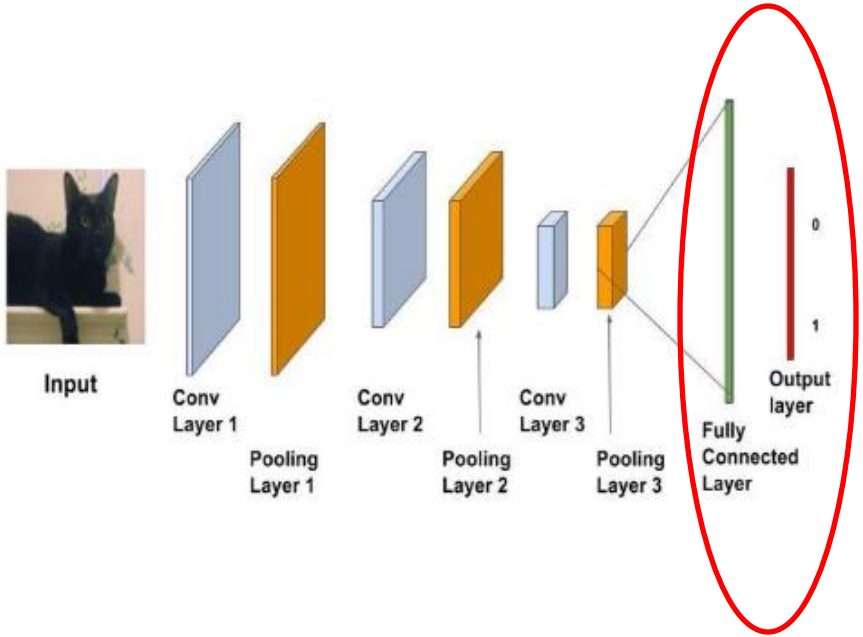


Transfer Learning



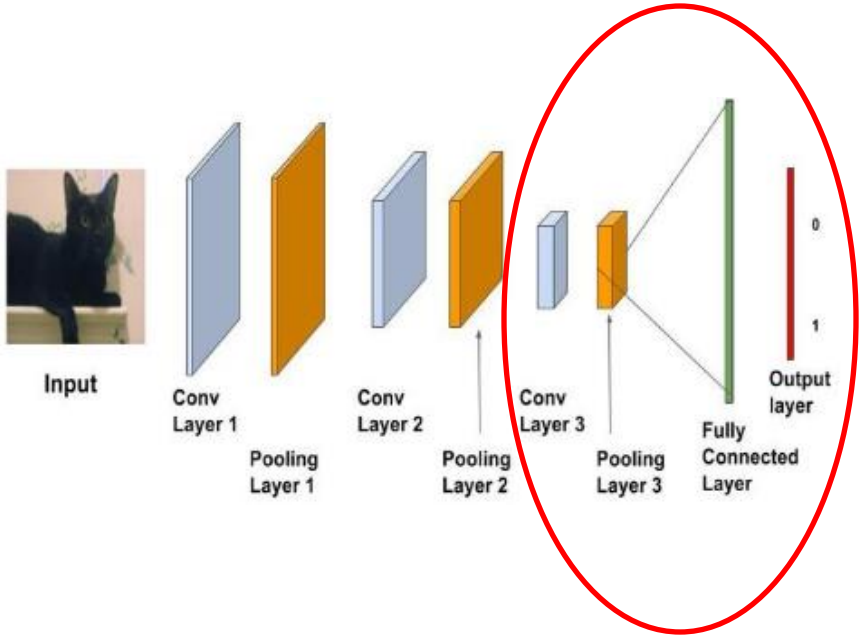
- SVM
- Random Forest
- etc.

Fine-tuning



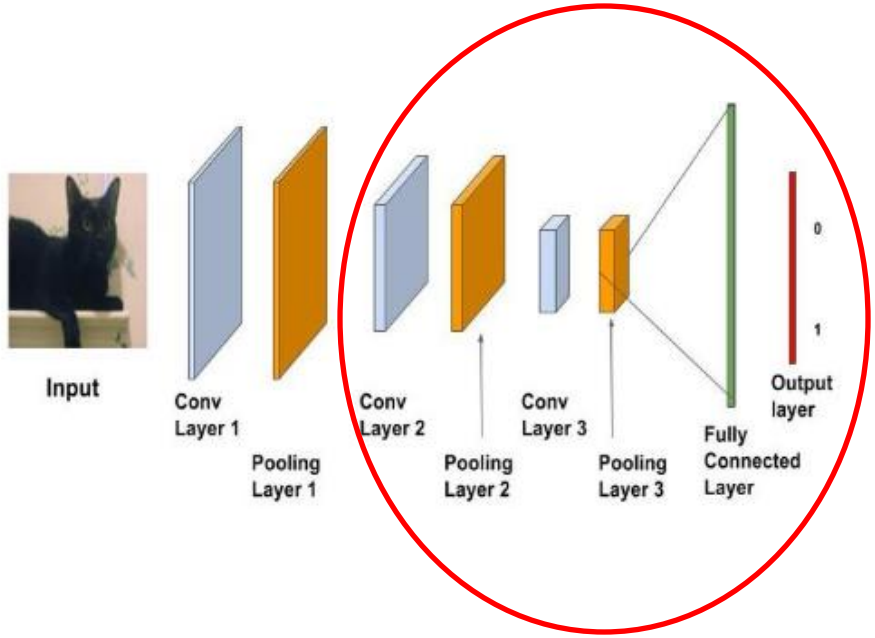
Shallow-tuning

Fine-tuning



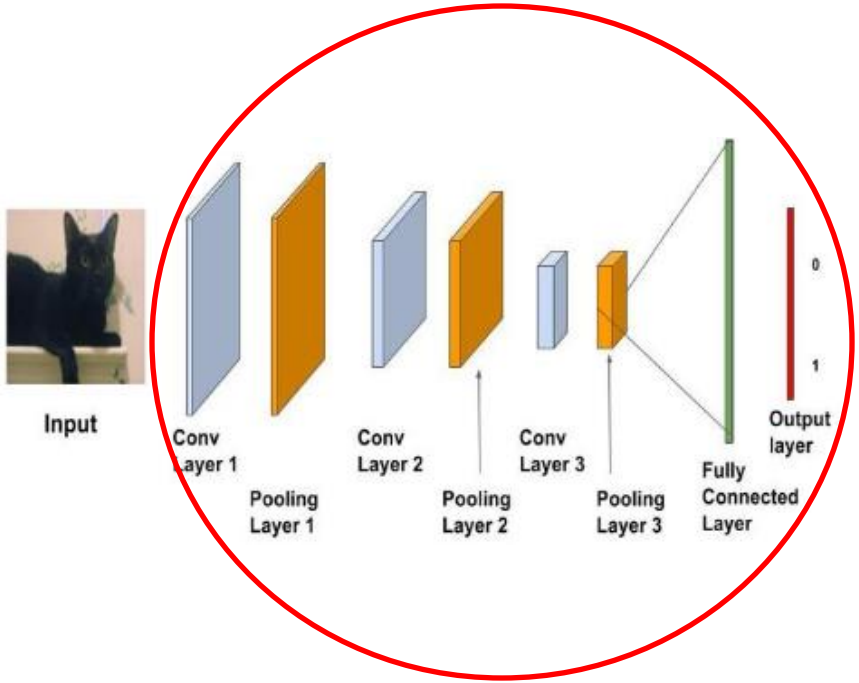
Deep-tuning

Fine-tuning



Deep-tuning

Fine-tuning



From scratch

Practical Protocol

In practise:

1. **Modify only the last layer** to have the same number of outputs as classes(**baseline**).
2. **Modify the top model** and **re-train** the **classifying stage** (FC layers) (**“shallow tuning”**).
3. **Re-train convolutional blocks** (**“deep tuning”**). In each iteration one more, starting from the output.

Tips depending on our dataset

- If it is **small and similar to the original**: transfer learning (SVM for example). It helps to prevent overfitting.
- If it is **large and similar to the original**: as we have more data we probably won't incur in over-fitting, so we can do fine-tuning with more confidence.

Practical protocol

Tips depending on our dataset

- If it is **small and very different** from the original: transfer learning but with characteristics of layers before the last convolutional.
- If it is **large and very different** from the original: train the network from scratch. (*from scratch*).

Note the possible restrictions of pre-trained models. For example, they may require a **minimum image size**. In addition, when re-training networks, **learning rates** are usually chosen **lower** than if we do it from scratch, since we start from an initialisation of weights that is assumed to be good.

Contents

Image DataGenerator class

- Keras has implemented functionalities to facilitate/optimise data loading.
- It is possible to load the training data in RAM in batches (batch by batch) to perform the steps that make up a training step and not having to store the entire dataset in memory.
- **ImageDataGenerator** is nothing more than an object type that can apply or not certain transformations to the data being loaded by means of a series of methods of this object that facilitate the task.
- The ImageDataGenerator object does not store the data itself.
- The transformations that ImageDataGenerator allows serve to preprocess the data, rescale it or establish a validation partition, but its main function is that it implements the functionality to create synthetic image samples.

https://keras.io/api/preprocessing/image/#imagedatagenerator-class

Loading batches of data from disk

The ImageDataGenerator object includes a series of methods that facilitate the task of loading data from disk. These methods allow the data to be loaded gradually into memory (batch by batch) during the training process.

`flow_from_directory` method

```
ImageDataGenerator.flow_from_directory(
    directory,
    target_size=(256, 256),
    color_mode="rgb",
    classes=None,
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix="",
    save_format="png",
    follow_links=False,
    subset=None,
    interpolation="nearest",
)
```

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Loading batches of data from disk

The ImageDataGenerator object includes a series of methods that facilitate the task of loading data from disk. These methods allow the data to be loaded gradually into memory (batch by batch) during the training process.

	A	B	C	D	E
1	image_name	Partition	COV	C	N
2	Caso_12_M_58_PA_DR_31BA3780-2323-493F-8AED-62081B9C383B.png	te	1	0	0
3	Caso_12_M_58_PA_DR_F2DE909F-E19C-4900-92F5-8F435B031AC6.png	tr	1	0	0
4	Caso_13_F_61_PA_DR_F051E018-DAD1-4506-AD43-BE4CA29E960B-958x1024.png	tr	1	0	0
5	Caso_14_M_50_PA_DR_1312A392-67A3-4EBF-9319-810CF6DA5EF6-1007x1024.png	tr	1	0	0
19	Caso_34_M_47_PA_DR_E63574A7-4188-4C8D-8D17-9D67A18A1AFA-1024x993.png	te	1	0	0
50	Caso_7_M_43_PA_DR_E1724330-1866-4581-8CD8-CEC9B8AFEDDE-1024x839.png	te	1	0	0
51	Caso_8_M_67_PA_DR_8FDE8DBA-CFBD-4B4C-B1A4-6F36A93B7E87.png	te	1	0	0
52	Caso_8_M_67_PA_DR_9C34AF49-E589-44D5-92D3-168B3B04E4A6.png	dev	1	0	0
53	Caso_9_F_73_PA_DR_72897418-E68C-47CA-B8F9-8EBAD8073476.png	dev	1	0	0
54	1000352387011846471720155937856633456 24_vb6v1o.png	tr	0	1	0
55	1000691030687536883475220935612068414 48_7197k3.png	dev	0	1	0
56	1000818202313855373970797295912664366 94_8o3uj2.png	tr	0	1	0
57	1001719840159941935625816910341180236 29_370g35.png	tr	0	1	0
58	1001892766955242748728120563805421040 60_5nnxx1.png	te	0	1	0
59	1002083827767486737014232995576939765 04_7znfv5.png	tr	0	0	1
60	1003056383150582931946274882075140984 68_p8ea27.png	tr	0	1	0

`flow_from_dataframe` method

```
ImageDataGenerator.flow_from_dataframe(
    dataframe,
    directory=None,
    x_col="filename",
    y_col="class",
    weight_col=None,
    target_size=(256, 256),
    color_mode="rgb",
    classes=None,
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix="",
    save_format="png",
    subset=None,
    interpolation="nearest",
    validate_filenames=True,
    **kwargs
)
```

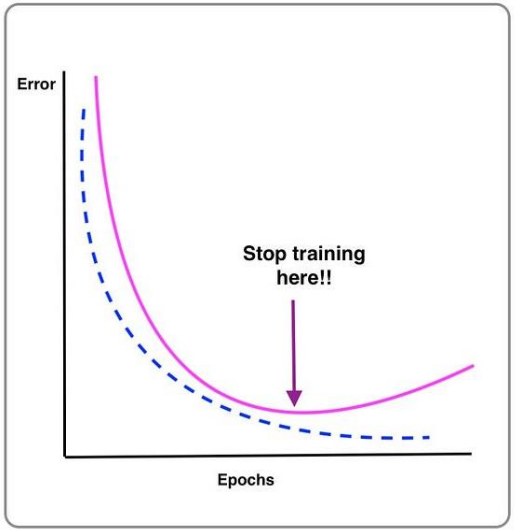
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

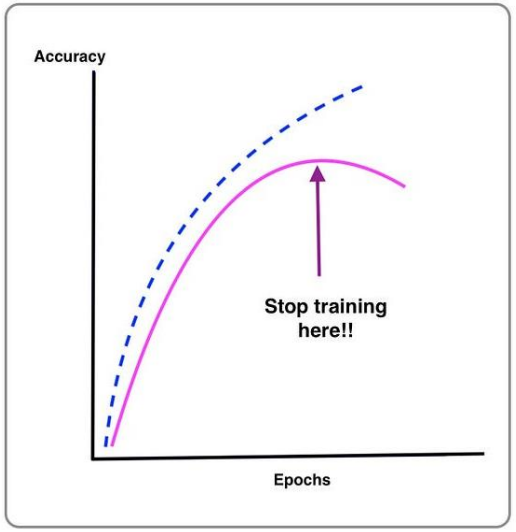
Data Augmentation

- **Objetive:** Increase the number of images in the training set to **combat overfitting**.
- How?
  - Rotation,
  - Re-scale,
  - Translation,
  - Zoom,
  - Flips (Horizontal and Vertical),
  - Changes in color.

- This is done **in real time, during training**. It is not necessary to save the augmented images.
- The **new images** take the **class of the source image**, so the transformations cannot be too exaggerated, let alone cause them to resemble another class.

Early Stopping






```
keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
    start_from_epoch=0,
)
```

https://keras.io/api/callbacks/early_stopping/

Data augmentation


```
# Creating an instance of ImageDataGenerator class
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   rotation_range = 0.2,
                                          zoom_range = 0.2,
                                 horizontal_flip = True,
                                          vertical_flip = True)

# Creating an instance of ImageDataGenerator class
validation_generator = validation_datagen.flow_from_directory(validationDirectory,
                                                              target_size=(img_width, img_height),
                                                              batch_size=batch_size,
                                                              class_mode='categorical')
```

Source: https://medium.com/towards-data-science/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085)

Reduce learning rate

Step decay

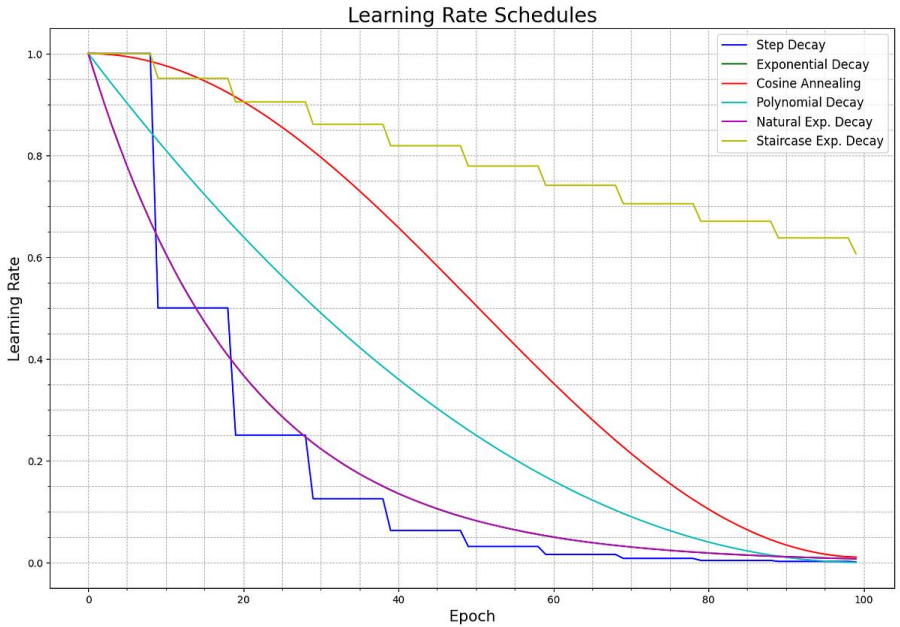
$$lr = lr_0 \cdot d^{(\text{floor}(1+\text{epoch})/s)}$$

Exponential decay

$$lr = lr_0 \cdot e^{-k \cdot \text{epoch}}$$

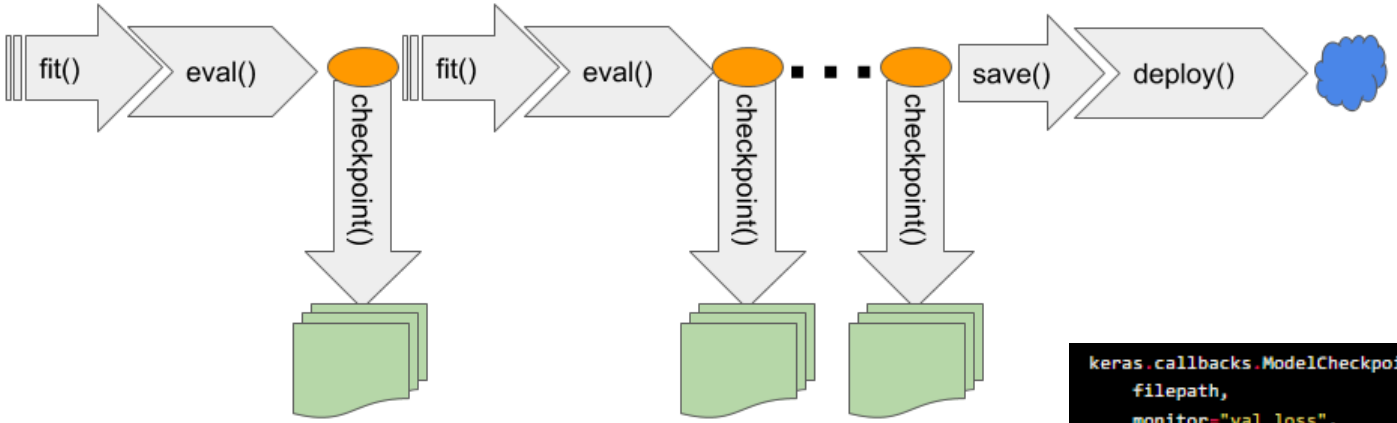
Cosine Annealing

$$lr = lr_{\min} + 0.5 \cdot (lr_{\max} - lr_{\min}) \cdot \left(1 + \cos\left(\frac{\text{epoch}}{\text{max\_epochs}} \cdot \pi\right)\right)$$



https://keras.io/api/callbacks/learning_rate_scheduler/

Model checkpoint




```
keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    initial_value_threshold=None,
)
```

https://keras.io/api/callbacks/model_checkpoint/