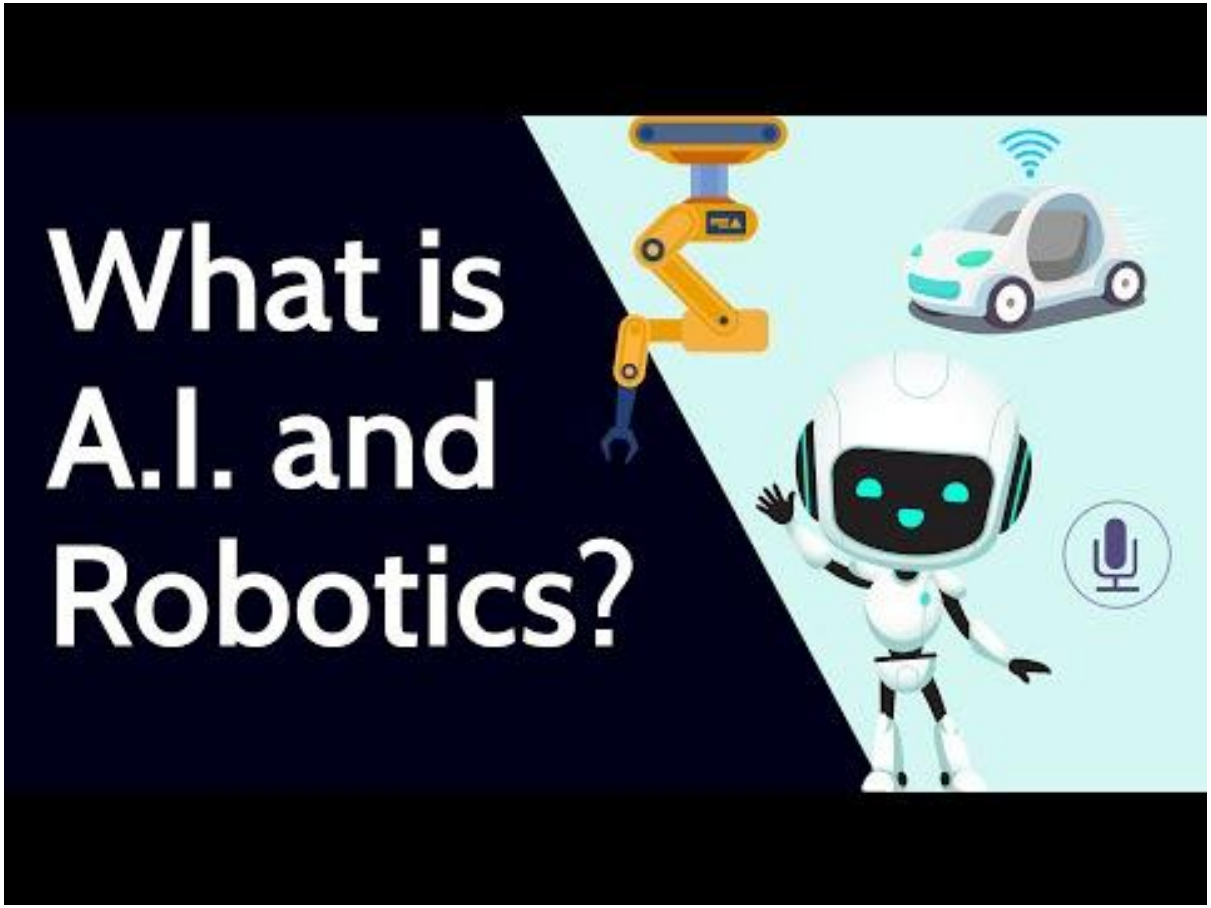

What is
A.I. and
Robotics?

Introduction to machine learning and artificial vision

Speaker: Eng. Giulio Mattera

mail:giulio.mattera@unina.it

Agenda

- Rule based algorithm vs machine learning
- Machine learning technique
  - PCA
  - k means
  - k-Nearest Neighbors
  - Single Layer Perceptron

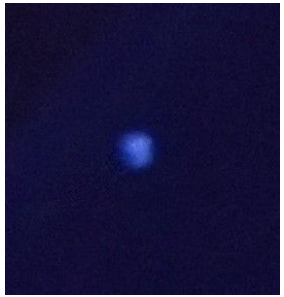
- Artificial vision
- Classical CV
  - How use filtering for image manipulation
- Deep neural network for computer vision
  - Basic theory
  - State of the art architecture
  - Transfer learning

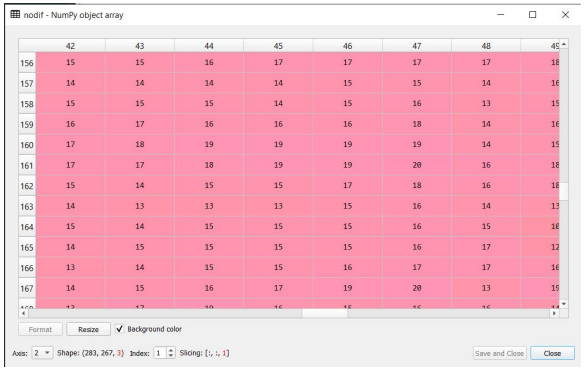
- Universal approximation theorem
- Modeling a shallowNet in Python
- Deep learning
  - Backpropagation
  - Gradient descent
  - How measuring performance
  - Hyper parameters tuning

- What's next? An quick intro to:
  - Recurrent neural networks
  - Reinforcement learning
- Open issue
- Introduction to AI framworks

- DNN Modeling with keras
- Demos
  - ShallowNet with TF
  - Welding defect detection with TF
  - Transfer learning for object classification
- DNN in MATLAB : Run or walking classifier

An example of non-ML algorithm (I)

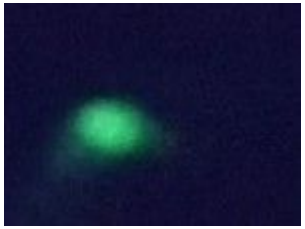


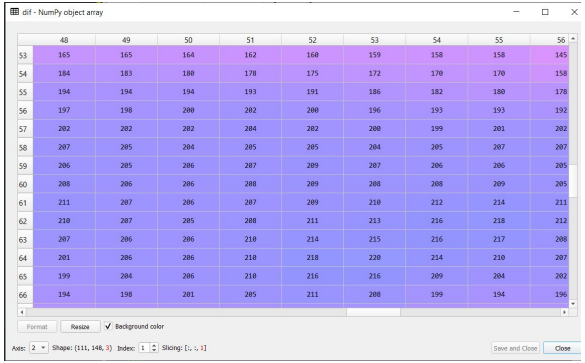


$$\mu_{blue} = 50.18, \sigma_{blue} = 38{,}46$$

$$\mu_{green} = 14{,}78, \sigma_{green} = 53{,}46$$

$$kurtosis\ blue\ mean = 0{,}69$$





$$\mu_{blue} = 55.11, \sigma_{blue} = 185.85$$

$$\mu_{green} = 24{,}34, \sigma_{green} = 954{,}39$$

$$kurtosis\ blue\ mean = 0{,}46$$

An example of non-ML algorithm (II)

An example of non-ML algorithm (III)

$$\mu_{blue} = 63, \sigma_{blue} = 127$$

$$\mu_{green} = 17, \sigma_{green} = 196$$

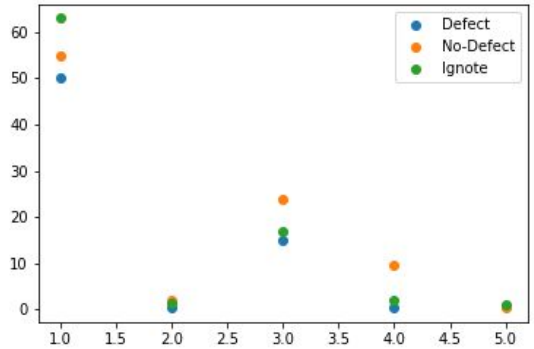
$$kurtosis\ blue\ mean = 1$$

An example of non-ML algorithm (IV)

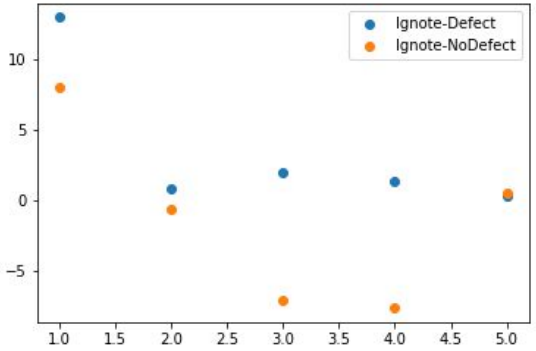
$$\mu_{blue} = 63, \sigma_{blue} = 127$$

$$\mu_{green} = 17, \sigma_{green} = 196$$

$$kurtosis\ blue\ mean = 1$$



$$d_{euc\ ignote-defect} = 17{,}63$$



$$d_{euc\ ignote-nodefect} = 6{,}62$$

An example of non-ML algorithm (V)

```python
import cv2
import glob
import numpy as np
import os

def ImgDataset(formato, path):

    """
    Import, resize and normalize images from a folder

    """
    dataPath = os.path.join(path)
    X_data = []
    files = glob.glob (dataPath+r"/*."+formato)
    for myFile in files:
        image = cv2.imread(myFile)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = image/255
        image = cv2.resize(image, (225, 225))
        X_data.append (image)
    X_data = np.array(X_data)

    return X_data

def extractFeatures(Dataset):

    mu = []
    var = []

    for i in range(Dataset.shape[0]):
        img = Dataset[i]
        mu.append(img.mean())
        var.append(img.var())

    feat = np.array([mu, var])

    return feat
```

Genera dataset caricando immagini da cartella

```python
import numpy
import my_lib

SAMPLE_IMG;
feat_sample = extractFeatures(SAMPLE_IMG)
d = numpy.linalg.norm(feat_defect - feat_sample)

if d < THR:
    sample = DEFECT
else:
    sample = OK
```

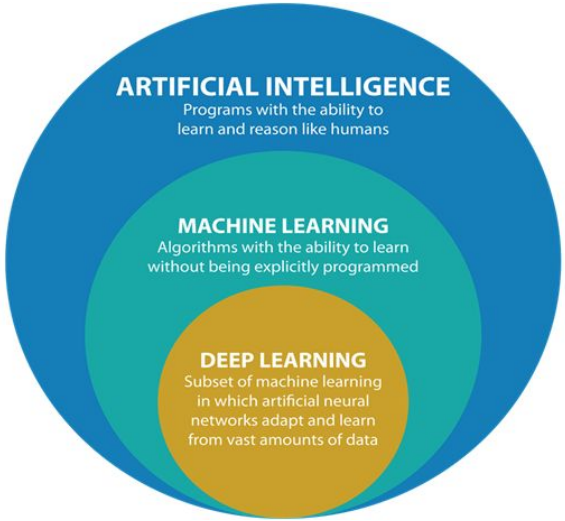
Estrai features dal dataset

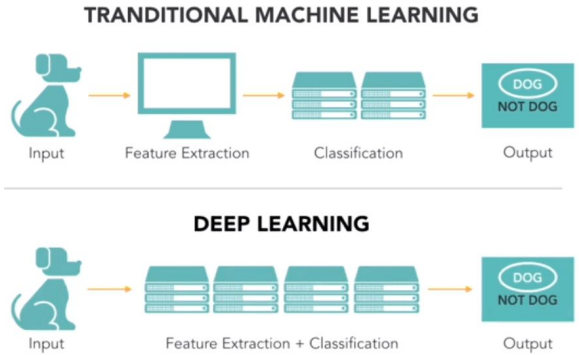
An example of non-ML algorithm (V)

- High interpretability
- Hard to find the best algorithm
- ...

Machine learning: definition (I)

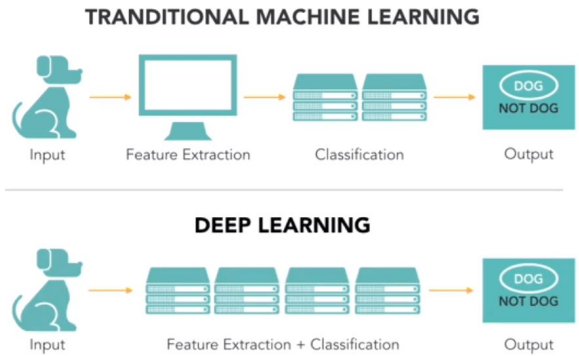
Machine learning is a set of optimization algorithm that allow code to learn patterns from data without being explicity programmed and is a child of **Artificial Intelligence.**

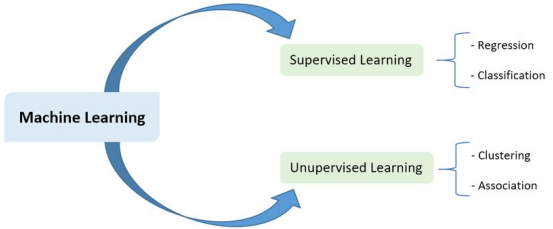




Machine learning: definition (II)

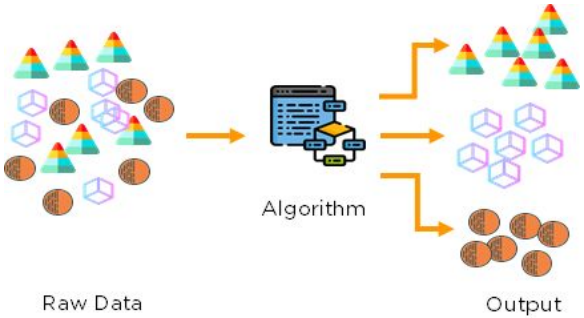
Machine learning is a set of optimization algorithm that allow code to learn patterns from data without being explicity programmed and is a child of **Artificial Intelligence.**





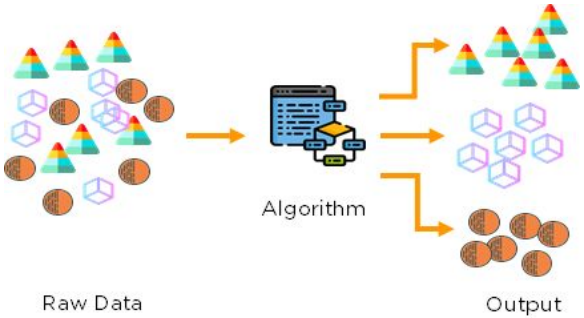
Unsupervised learning (I)

- The algorithm is shown a set of data with **no labels.**
- The goal is to **discover some hidden structure** contained in the data and provide a more compact representation



Unsupervised learning (II)

- The algorithm is shown a set of data with **no labels.**
- The goal is to **discover some hidden structure** contained in the data and provide a more compact representation



The typical applications are:

- Dimensional reduction
- Clustering
- Anomaly detection

Machine learning: Principal Component Analysis (I)

A features is a characteristic of the dataset:

- Age of students
- Height and weight
- Eye's color
- Mean of blue intensity in an image
- Etc..

These features can be visualized in 2D/3D plots.

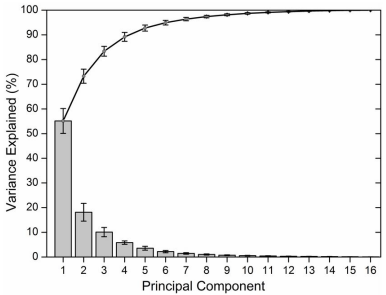
Machine learning: Principal Component Analysis (II)

A features is a characteristic of the dataset:

- Age of students
- Height and weight
- Eye's color
- Mean of blue intensity in an image
- Etc..

These features can be visualized in 2D/3D plots.

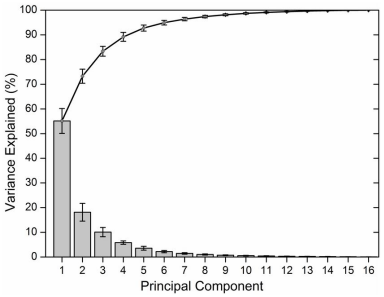
The results of a **PCA** could be summarized in a Pareto diagram:



- Features that explain better the variance of dataset could be used for any future task
- The features could be re-plotted in features space
- We could find clusters of data in features space that indicates their correlations

Machine learning: Principal Component Analysis (III)

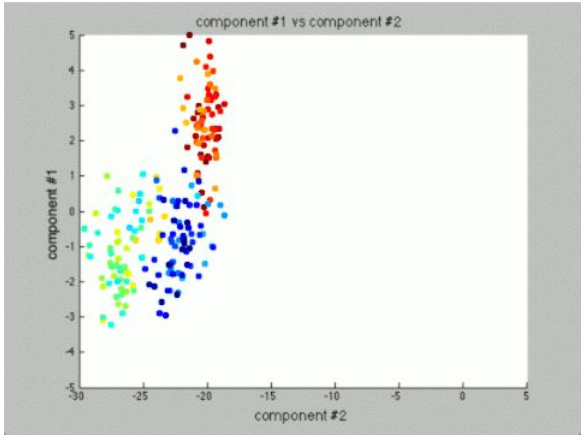
The results of a **PCA** could be summarized in a Pareto diagram:



- Features that explain better the variance of dataset could be used for any future task
- The features could be re-plotted in features space
- We could find clusters of data in features space that indicates their correlations

```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
pca = PCA(n_components=2) #components to keep
pca.fit(X)
print(pca.explained_variance_ratio_)
```

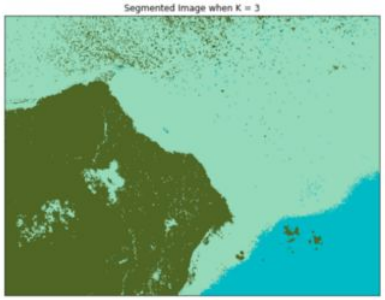
[0.9924, 0.0075]

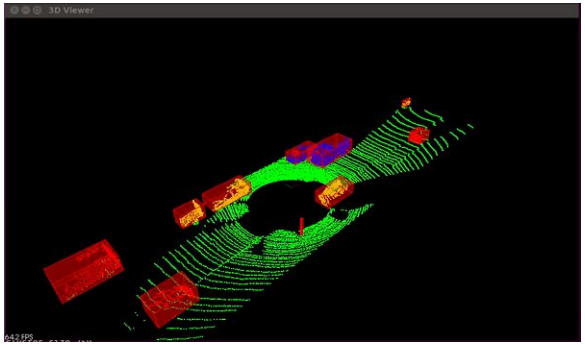


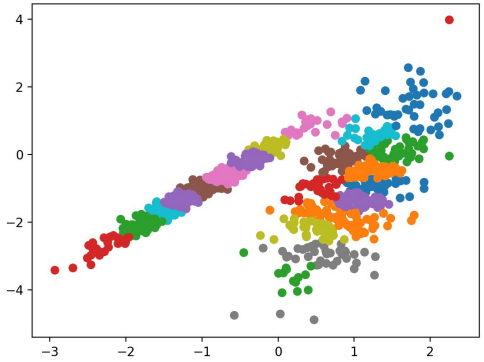
Clustering

Some common applications for clustering include the following:

- Social network analysis
- Search result grouping
- Image segmentation
- Anomaly detection
- Unknown labels generation







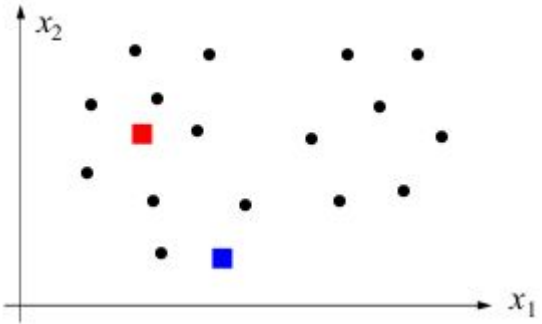
k-means (I)

It is an iterative algorithm used to partition a set of N samples into K clusters. Each cluster is identified by its centroid μ .
It is computationally complex (NP-hard), because is based on **brute force method** but normally converges quickly to a local optimum.

k-means (II)

It is an iterative algorithm used to partition a set of N samples into K clusters. Each cluster is identified by its centroid μ .
It is computationally complex (NP-hard), because is based on **brute force method** but normally converges quickly to a local optimum.

- It starts by assigning K random centroids



k-means (III)

It is an iterative algorithm used to partition a set of N samples into K clusters. Each cluster is identified by its centroid μ .
It is computationally complex (NP-hard), because is based on **brute force method** but normally converges quickly to a local optimum.

- It starts by assigning K random centroids
- Then, each sample is assigned the cluster whose centroid has the smallest distance. Usually **Minkowski** metrics is used:

$$dist = \left(\sum_{k=1}^{n} | p_k - q_k |^r \right)^{\frac{1}{r}}$$

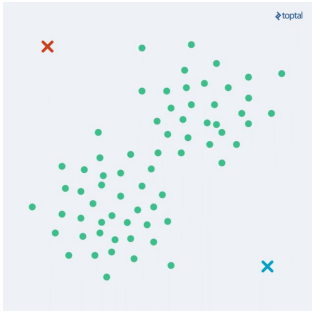
k-means (IV)

It is an iterative algorithm used to partition a set of N samples into K clusters. Each cluster is identified by its centroid μ .
It is computationally complex (NP-hard), because is based on **brute force method** but normally converges quickly to a local optimum.

- It starts by assigning K random centroids
- Then, each sample is assigned the cluster whose centroid has the smallest distance. Usually **Minkowski** metrics is used.

$$dist = \left(\sum_{k=1}^{n} | p_k - q_k |^r\right)^{\frac{1}{r}}$$

- When all the samples are assigned to a cluster, a new centroid is calculated and the process repeat until convergence (no changes) or max number of iteration are reached.



k-means (V)

Input: Number of clusters: K
Set of points: $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, where $\boldsymbol{x}_i \in \mathbb{R}^{dim}$

Output: Cluster centroids: $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$, where $\boldsymbol{\mu}_k \in \mathbb{R}^{dim}$
Cluster assignment: $\boldsymbol{c} = \{c_1, \ldots, c_N\}$

K_means(K, $\boldsymbol{x}$, $\boldsymbol{\mu}$, $\boldsymbol{c}$)
{

Initialize the cluster centroids coordinates with random numbers;

repeat

1. Assign each point $\boldsymbol{x}_i \in X$ the index of the cluster with the closest centroid:

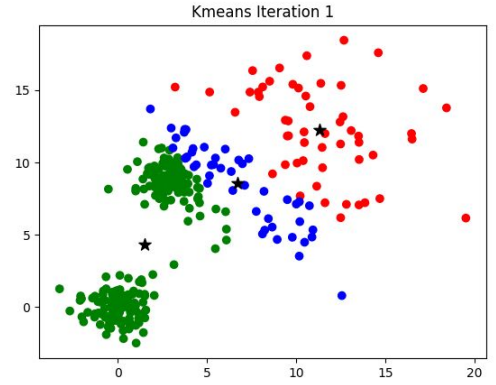
$$\forall i = 1, \ldots, N \quad c(i) = \arg\min_k dist(\boldsymbol{\mu}_k, \boldsymbol{x}_i);$$

2. Update each cluster centroid with the mean coordinates of the points in each cluster:

$$\forall k = 1, \ldots, K, \forall d = 1, \ldots, dim \quad \boldsymbol{\mu}_{kd} = \sum_{i:\, c(i)=k} \boldsymbol{x}_{id} \Big/ \sum_{i:\, c(i)=k} 1$$

until no centroid changes

}

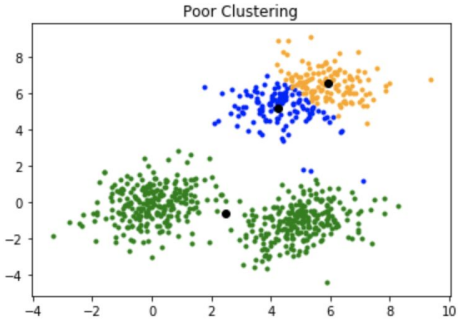


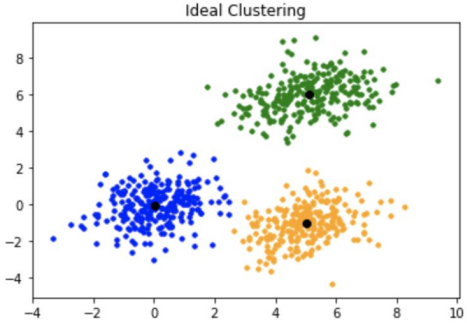

```
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0], ... [10, 2], [10, 4], [10, 0]])

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

print(kmeans.labels_)
```

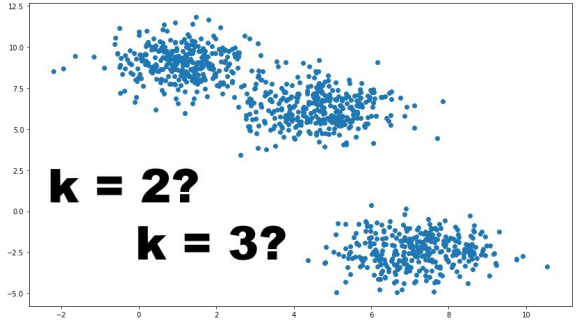
[1, 1, 1, 0, 0, 0]

k-means (I)

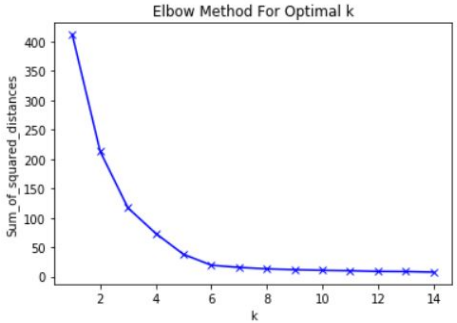
- A better way to initialize the centroids is to put them in some random points of that dataset.
- How choose K?





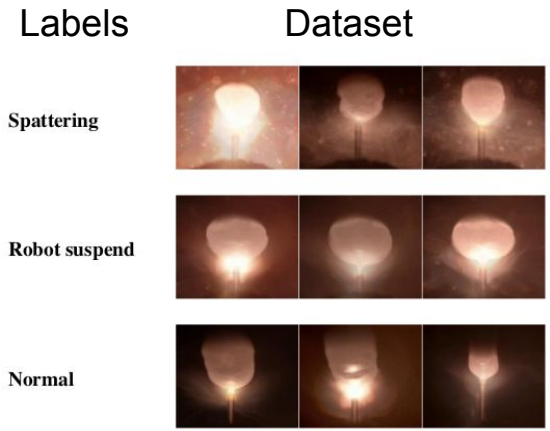


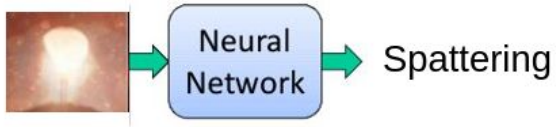
Elbow method:



Supervised learning (I)

Given a **training set,** composed by data and label, the objective of an supervised learning algorithm is to **learn** from data to solve problem of **classification and regression.**



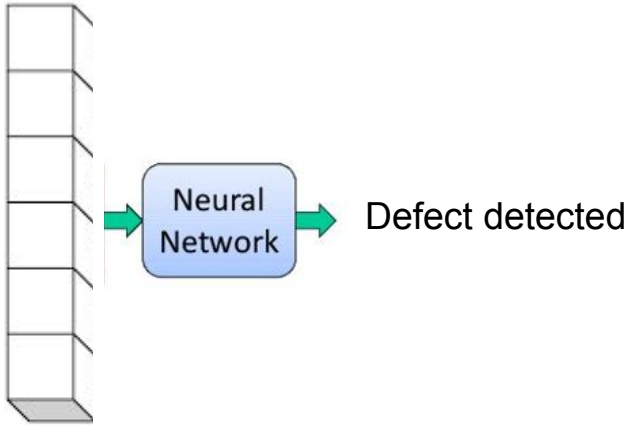


Supervised learning (II)

Given a **training set,** composed by data and label, the objective of an supervised learning algorithm is to **learn** from data to solve problem of **classification and regression.**

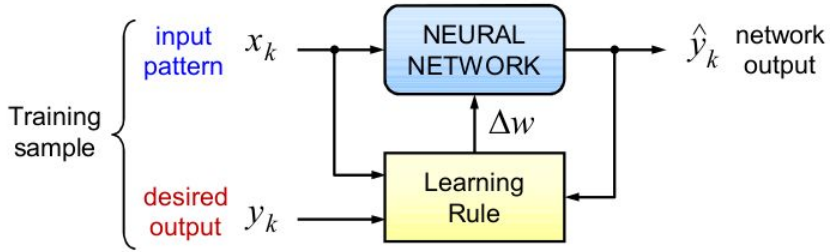
Dataset

												Labels
290	310	300	290	247	260	273	29	28,8	380	400	702	1
290	310	300	290	253	272,5	292	29	28,80	380	400	846	1
290	310	300	290	249	260,5	272	29	28,80	380	400	678	1
290	310	300	290	245	262	279	29	28,8	380	400	780	1
290	310	300	290	256	265	274	29	28,8	380	400	660	1
290	310	300	290	253	268	283	29	28,80	380	400	702	1
140	150	140	140	131	138	145	19	28,8	140	160	552	1
210	230	220	210	223	243	263	24	28,8	250	270	342	1
290	310	300	290	252	269	286	29	28,8	380	400	276	1
290	310	300	290	257	269,5	282	29	28,8	380	400	330	1
140	160	150	140	138	151	164	19	31,30	140	160	552	1
210	230	220	210	236	254	272	24	31,30	250	270	588	1
290	310	300	290	228	248	268	29	28,8	380	400	402	0
140	160	150	140	150	168	186	19	31,30	140	160	678	1
210	230	220	210	233	251	269	24	31,30	250	270	732	1
290	310	300	290	226	245,5	265	29	28,8	380	400	420	0
140	160	150	140	212	243	274	19	31,30	140	160	324	1
210	230	220	210	229	251,5	274	24	31,30	250	270	1500	0
290	310	300	290	244	266,5	289	29	28,8	380	400	264	1
140	160	150	140	220	243	266	19	31,30	140	160	624	1
210	230	220	210	248	278	308	24	31,30	250	270	420	1
290	310	300	290	238	251	264	29	28,8	380	400	522	0
140	160	150	140	226	248,5	271	19	31,30	140	160	624	1
210	230	220	210	246	272	298	24	31,30	250	270	468	1
290	310	300	290	231	244	257	29	28,8	380	400	426	0
140	160	150	140	252	272,5	293	19	28,8	140	160	204	1
210	230	220	210	237	253,5	270	24	28,8	250	270	198	1
290	310	300	290	227	247,5	268	29	28,8	380	400	264	0
290	310	300	290	243	256	269	29	28,8	380	400	618	1
140	160	150	140	101	126,5	152	19	21,4	140	160	714	1
140	160	150	140	129	138	147	19	20,9	140	160	666	1
140	160	150	140	128	140	152	19	21,1	140	160	300	1
140	160	150	140	129	137,5	146	19	21,1	140	160	318	1
140	160	150	140	117	130,5	144	19	21,00	140	160	390	1
140	160	150	140	141	151	161	19	21,7	140	160	540	1
290	310	300	290	255	264,5	274	29	31,3	380	400	594	1
290	310	300	290	256	272	288	29	31,3	380	400	846	1
290	310	300	290	236	255,5	275	29	31,3	380	400	246	1
290	310	300	290	256	266	276	29	31,3	380	400	258	1
290	310	300	290	248	269	290	29	28,8	380	400	192	1
140	160	150	140	119	143,5	168	19	21,00	140	160	150	1
210	230	220	210	264	286,5	309	24	30,8	250	270	336	1

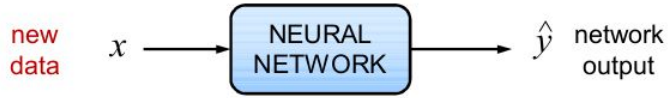


Supervised learning (III)

Training phase:

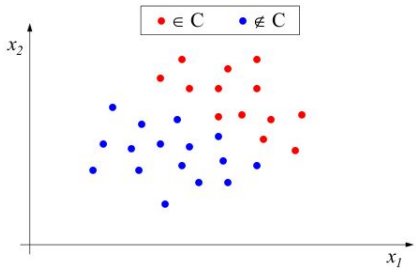


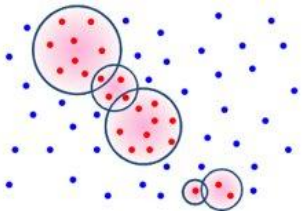
Inference:



Classification

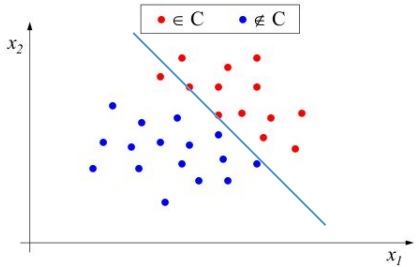
We want to assign the correct labels from a prior knowledge.



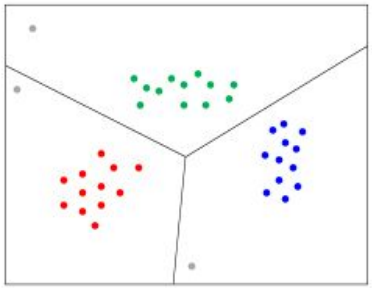


RBFNet classifier

The idea could be quite similar to clustering, but in this case we have a prior knowledge that networks use to learn.



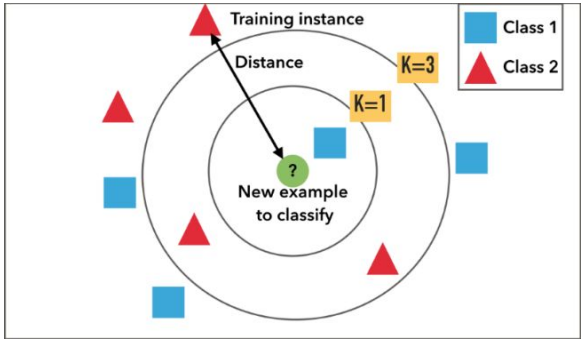
Binary classifier



Multiclass classifier

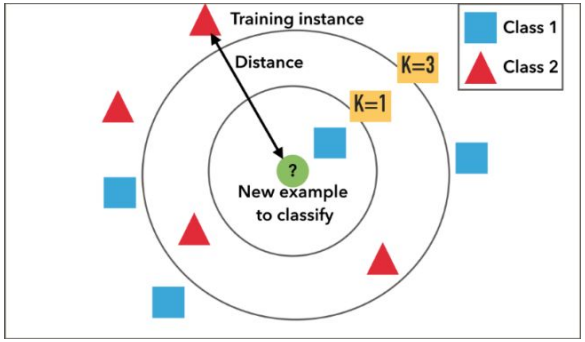
k-Nearest Neighbors (I)

The principle behind nearest neighbor methods is quite simple. We want to classify a new input, based on his similarity of a defined neighbor (number of samples). The similarity is generally measured with standard Euclidean distance metrics.



k-Nearest Neighbors (II)

The principle behind nearest neighbor methods is quite simple. We want to classify a new input, based on his similarity of a defined neighbor (number of samples). The similarity is generally measured with standard Euclidean distance metrics.



- This algorithm is expensive due to brute force method and the necessary of sort the all distances calculated to find which sample is closest to new one.
- The label of new sample is equal to the most frequent in selected neighbor

k-Nearest Neighbors (III)

Customer	Age	Loan	Default	Euclidean distance
John	25	40000	N	1,02,000.00
Smith	35	60000	N	82,000.00
Alex	45	80000	N	62,000.00
Jade	20	20000	N	1,22,000.00
Kate	35	120000	N	22,000.00
Mark	52	18000	N	1,24,000.00
Anil	23	95000	Y	47,000.01
Pat	40	62000	Y	80,000.00
George	60	100000	Y	42,000.00
Jim	48	220000	Y	78,000.00
Jack	33	150000	Y	8,000.01
Andrew	**48**	**142000**	**?**	

What happen with k=3 or k=5?

k-Nearest Neighbors (IV)

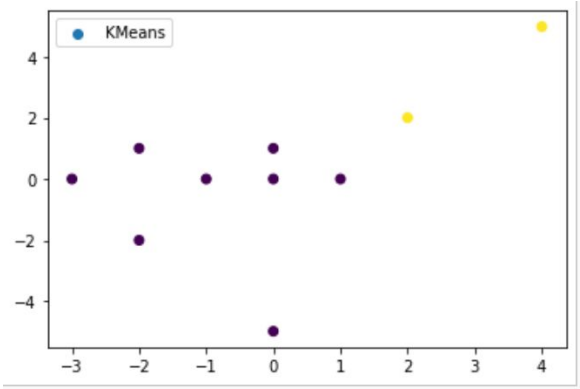
Customer	Age	Loan	Default	Euclidean distance
John	25	40000	N	1,02,000.00
Smith	35	60000	N	82,000.00
Alex	45	80000	N	62,000.00
Jade	20	20000	N	1,22,000.00
Kate	35	120000	N	22,000.00
Mark	52	18000	N	1,24,000.00
Anil	23	95000	Y	47,000.01
Pat	40	62000	Y	80,000.00
George	60	100000	Y	42,000.00
Jim	48	220000	Y	78,000.00
Jack	33	150000	Y	8,000.01
Andrew	**48**	**142000**	**?**	

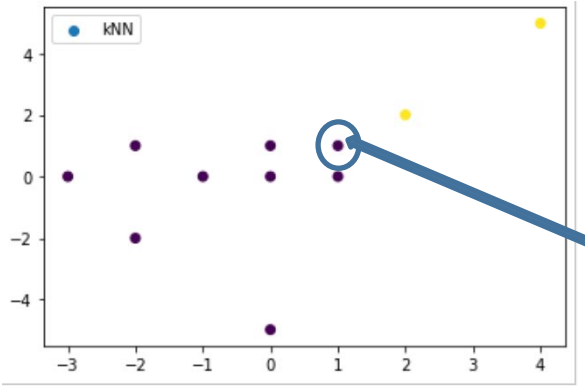
What happen with k=3 or k=5?

- Larger value correspond to smoother decision boundaries (low variance but high bias)
- A good starting point could be select k = sqrt(N)

How use unsupervised and supervised learning

```python
from sklearn import neighbors, cluster
import numpy as np
import matplotlib.pyplot as plt

#Create Dataset and a new point x to classify
samples = np.array([(0, 0), (0, 1), (1, 0), (4,5), (2,2), (-1,0), \
                    (-2,-2),(-3,0),(-2,1), (0,-5)])
x = np.array([1,1])
x = x.reshape(1, -1)

#Inizialization
K = 2

#Unsupervised classification to create lables
kmeans = cluster.KMeans(K, init = 'random', n_init = 5, \
                        max_iter = 10, tol = 1e-3,\
                        n_jobs=-1, algorithm='full').fit(samples)

y = kmeans.labels_
y = np.array(y)
#Supervised classification with kkN

kNN = neighbors.KNeighborsClassifier(n_neighbors = K, \
                                     algorithm = 'brute', \
                                     p = 2, n_jobs = -1).fit(samples,y)

y_pred = kNN.predict(x)

#Creating new dataset and plotting
Nsamples = np.concatenate((samples,x), axis=0)
Ny = np.concatenate((y,y_pred), axis=0)
plt.scatter(samples[:,0], samples[:,1], c=y, label='KMeans')
plt.legend()
```

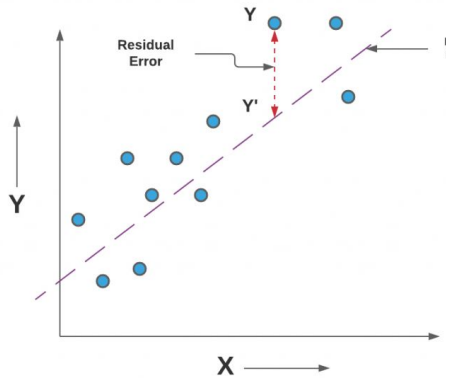




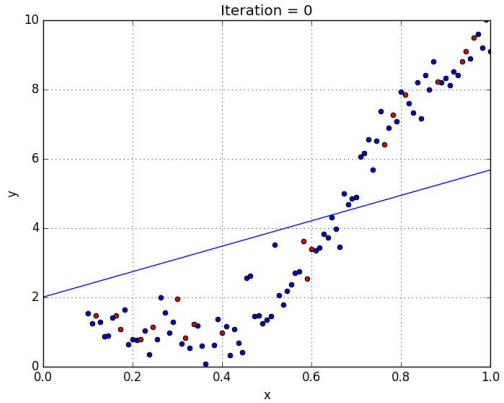
New one

Machine learning: regression

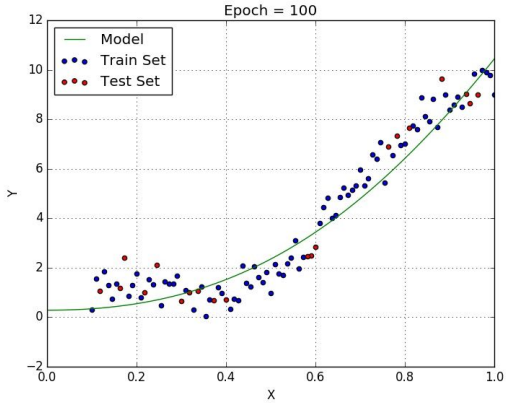
The purpose of regression is the same as classification, but the difference is that in this case we want to predict a continuous variable rather than a discrete one.



OLS/Statistical methods



Linear learning



Unknown model learning

Ordinary Last Square Estimator (I)

Suppose we have n observation x=[x1..xn] and p true values y=[y1..yp], we call general linear regression model:

$$y = Ax + B$$

Ordinary Last Square Estimator (II)

Suppose we have n observation x=[x1..xn] and p true values y=[y1..yp], we call general linear regression model:

$$y = Ax + B$$

Where A is the weights matrix and B the biases matrix. The goal is to find the values of A and B that minimizes the error:

$$E = y_{noto} - y_{predict}$$

Ordinary Last Square Estimator (III)

Suppose we have n observation x=[x1..xn] and p true values y=[y1..yp], we call general linear regression model:

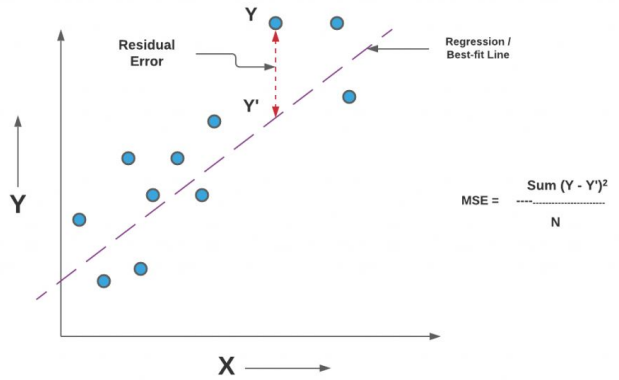
$$y = Ax + B$$

Where A is the weights matrix and B the biases matrix. The goal is to find the values of A and B that minimizes the error:

$$E = y_{noto} - y_{predict}$$

Or better, that minimizes the last square error:

$$MSE = \frac{e^2}{2}$$



Ordinary Last Square Estimator (IV)

Suppose we have n observation x=[x1..xn] and p true values y=[y1..yp], we call general linear regression model:

$$y = Ax + B$$

Where A is the weights matrix and B the biases matrix. The goal is to find the values of A and B that minimizes the error:

$$E = y_{noto} - y_{predict}$$

The Ordinary Last Square estimator for A is:

$$A = (x^T x)^{-1} x^T y_{noto}$$

OLS is the best estimator for A in the sense that allow the minimum mean and variance of solution.

Ordinary Last Square Estimator (V)

Suppose we have n observation x=[x1..xn] and p true values y=[y1..yp], we call general linear regression model:

$$y = Ax + B$$

Where A is the weights matrix and B the biases matrix. The goal is to find the values of A and B that minimizes the error:

$$E = y_{noto} - y_{predict}$$

The Ordinary Last Square estimator for A is:

$$A = (x^T x)^{-1} x^T y_{noto}$$

OLS is the best estimator for A in the sense that allow the minimum mean and variance of solution.

$$B = Ax - y_{predict}$$

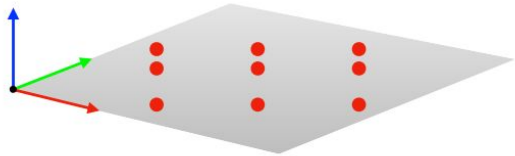
NB: exists only if there are at least as many measurements as the predicted variables:.

OLS: Plane estimation

```python
'''
The function estimates the plane
    in which a number of points in 3D space lie.

LastLast modified: 17/05/21 @ 15.57
Executed by: Giulio Mattera
Property: Giulio Mattera
'''

import numpy as np

def planeDetection(points):
    b = points[:,2]
    b = b.T
    x = points[:,0]
    y = points[:,1]
    ones = np.ones(points.shape[0])
    A=np.array([ones,x,y])
    A=A.T

    c_plane = np.linalg.inv(A.T.dot(A)).dot(A.T).dot(b)
    errors = b.T - A.dot(c_plane)
    residual = np.linalg.norm(errors)

    return c_plane,errors,residual

samples = np.random.rand(10, 3)

c, _, _ = planeDetection(samples)
#Executed code with numpy function to verify our code
A = np.array([np.ones(samples.shape[0]) ,samples[:,0], samples[:,1]])
coef, _, _, _ = np.linalg.lstsq(A.T, samples[:,2], rcond=None)
print('[INFO] The difference between 2 models is ', c-coef)
```

Equation of a plane in 3D:

$$z = a + bx + cy$$

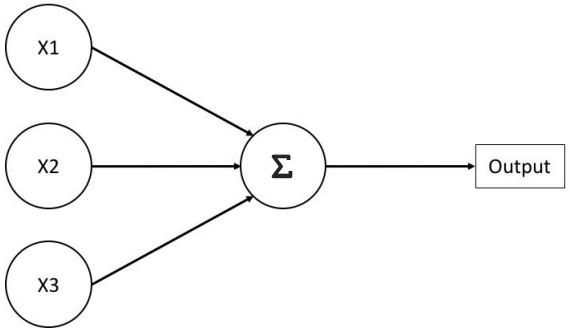


$$z_i = a + bx_i + cy_i$$

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} - \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = 0$$

$$e_i = \hat{z}_i - z_i$$

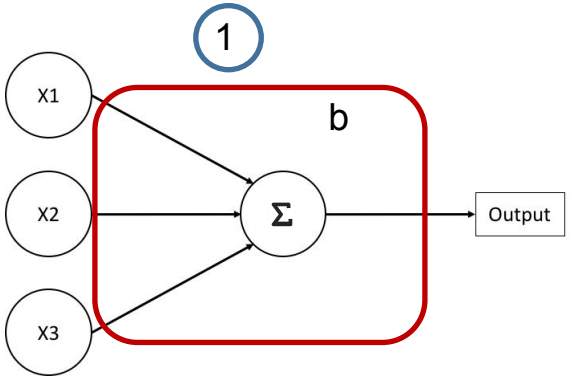
Introduction to neurons (I)



A neuron is a linear combination of inputs.

$$o = \sum_{i}^{n} w_i x_i$$

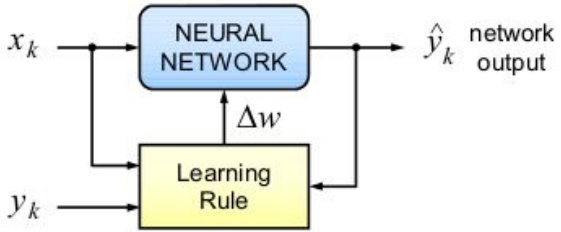
Introduction to neurons (II)



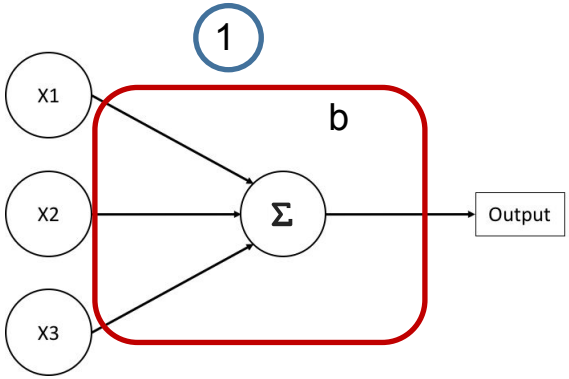
A neuron is a linear combination of inputs.

$$o = \sum_{i}^{n} w_i x_i$$

Where we could add a bias.



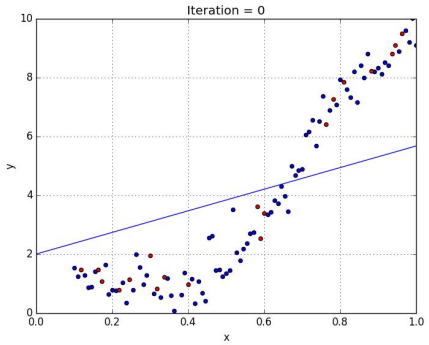
Introduction to neurons (III)

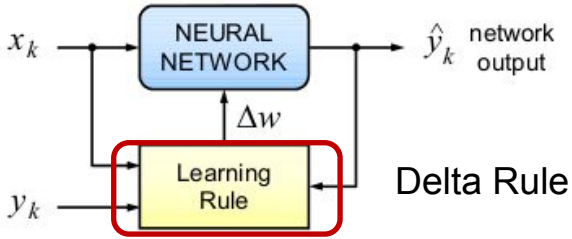


A neuron is a linear combination of inputs.

$$o = \sum_{i}^{n} w_i x_i$$

Where we could add a bias.

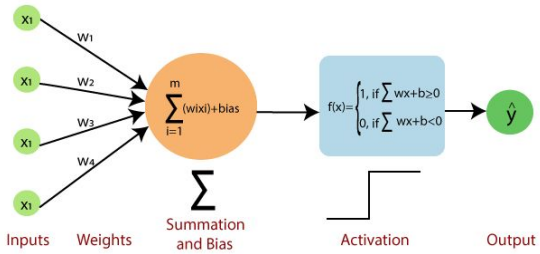




Delta Rule. Intuition from *Widrow* and *Hoff*

$$\Delta W = \alpha(t - \hat{y}) \sum x_i$$

Single Layer Perceptron : Rosenblatt ['57]



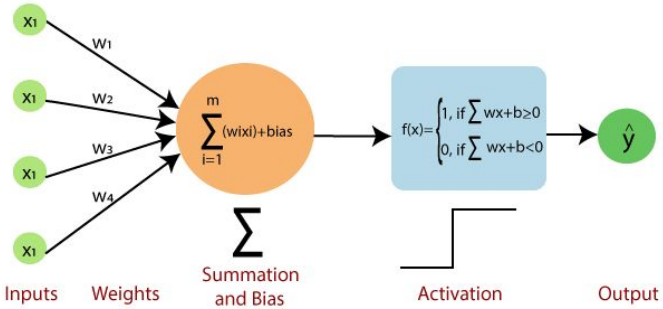
The Rosenblatt perceptron is used for classification task. For this purpose an **activation function** is used.

1. Fix a training set of M examples:

   $$TS = \{(\boldsymbol{x}_k, t_k),\ k = 1, \ldots, M\}$$

2. Initialize weights with random values $w_i \in [0,1]$
3. Take a pair $(\boldsymbol{x}_k, t_k)$ as input;
4. Compute the output y_k and the error $\delta = (t_k - y_k)$;
5. Update weights with the *Delta Rule*: $(\Delta \boldsymbol{w} = \eta \delta \boldsymbol{x})$;
6. Repeat the cycle from step 3, until all the outputs are correct, that is: $y_k = t_k \ \ \forall k \in [1,M]$

Single Layer Perceptron : Rosenblatt ['57]



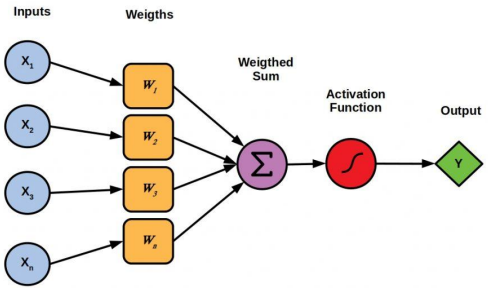
The Rosenblatt perceptron is used for classification task. For this purpose an **activation function** is used.

1. Fix a training set of M examples:
   $$TS = \{(\boldsymbol{x}_k, t_k),\ k = 1, \ldots, M\}$$
2. Initialize weights with random values $w_i \in [0,1]$
3. Take a pair $(\boldsymbol{x}_k, t_k)$ as input;
4. Compute the output y_k and the error $\delta = (t_k - y_k)$;
5. Update weights with the *Delta Rule*: $(\Delta \boldsymbol{w} = \eta \delta \boldsymbol{x})$;
6. Repeat the cycle from step 3, until all the outputs are correct, that is: $y_k = t_k \ \ \forall k \in [1,M]$

In 1969, Minsky & Papert showed the strongest limit of the Perceptron:

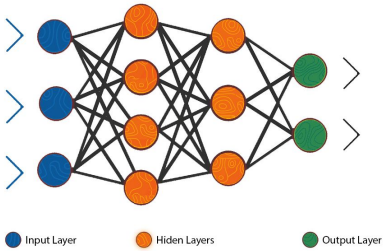
- to learn a classification, the problem must be linearly separable

Single Layer Perceptron : Limits and solution (I)

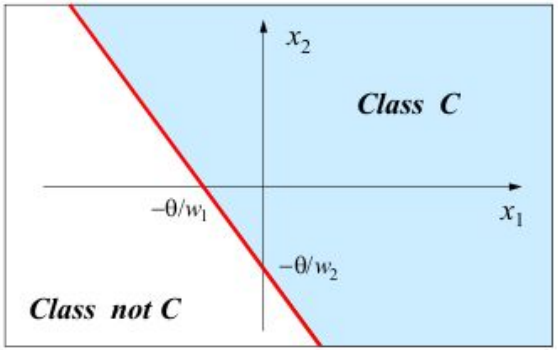


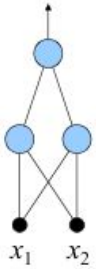
The problem could be solved:

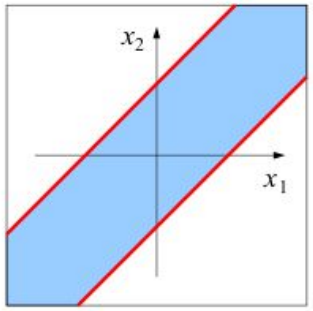
- Using a different activation function
- Using more hidden layers where all the neurons of a layer are connected to all the neurons of the next layer and there are no connections between neurons in the same layer and between non adjacent layers. (Rumelhart-Hinton-Williams, 1986)

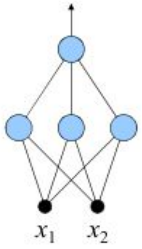


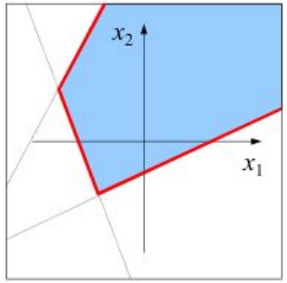
Single Layer Perceptron : Limits and solution (II)

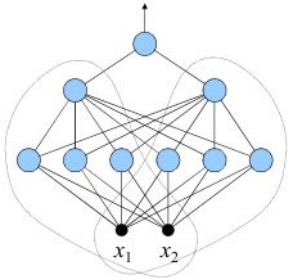


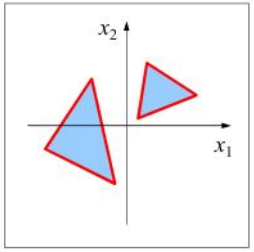










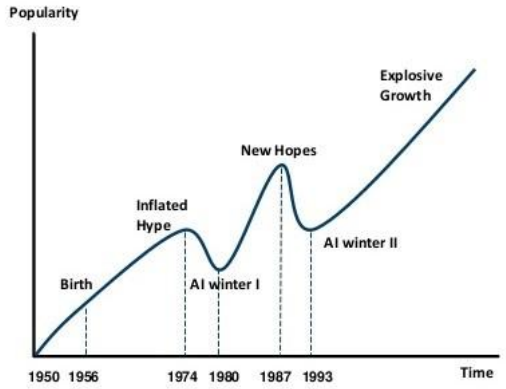


Single Layer Perceptron : Limits and solution (III)

To perform complex classifications, neurons must be non linear and must be organized on multiple layers.
Problems:

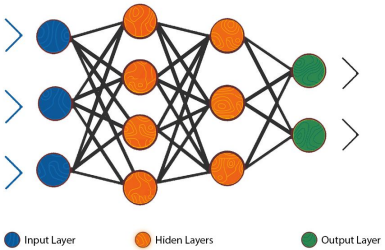
- How do we train a multi-layered non linear network?

This problem remained unsolved for many years, causing the so called “AI winter”.



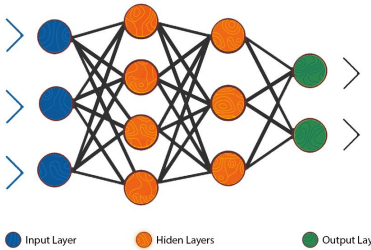
Multi layer perceptron (I)

When there are more then 1 hidden layer we tell about MLP. If hidden-layers > 3 we tell about **deep neural netwroks.**

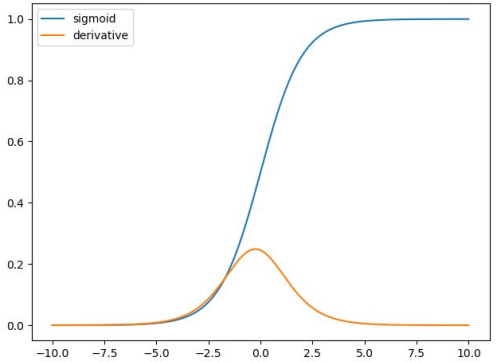


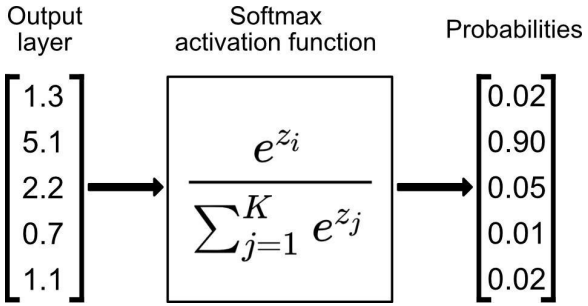
Multi layer perceptron (II)

When there are more then 1 hidden layer we tell about MLP. If hidden-layers > 3 we tell about **deep neural netwroks.**



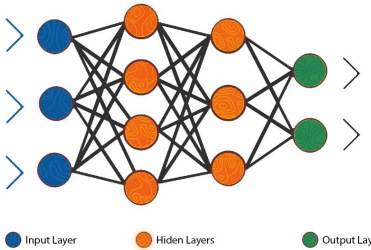
When the MLP solve a **classification** problem the activation function in the output layer is the sigmoid or softmax:



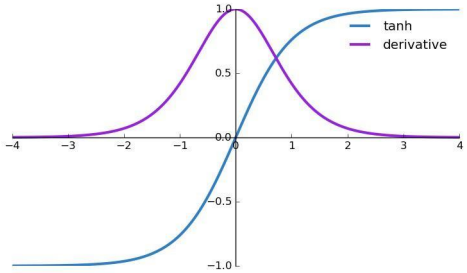


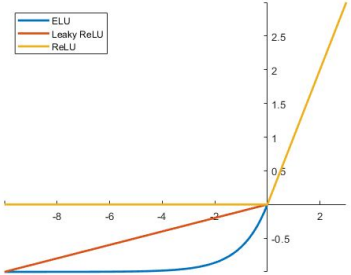
Multi layer perceptron (III)

When there are more then 1 hidden layer we tell about MLP. If hidden-layers > 3 we tell about **deep neural netwroks.**



When the MLP solve a **regression** problem the activation function in the output layer is the tanh or ReLu:





Agenda : Lesson 2

- Universal approximation theorem
- Modeling a shallowNet in Python
- Deep learning
  - Backpropagation
  - Gradient descent
  - How measuring performance
  - Hyper parameters tuning

Universal approximation theorem

In 1989, George Cybenko proved an important results known as the Universal Approximation Theorem.

A feed-forward network with a single hidden layer of neurons with a sigmoid activation function can approximate any continuous function.

In 1991, Kurt Hornik showed that this property does not depend on the specific activation function, but rather on the multilayer feed-forward architecture with non linear activations.
Although a single hidden layer is powerful enough to learn any function, in practice, multiple hidden layers can learn better.

Backpropagation (I)

Backpropagation (II)

1. Compute Loss

$$L_2 = \frac{1}{2}\sum(y_i - t_i)^2$$

Backpropagation (III)

1. Compute Loss

$$L_2 = \frac{1}{2}\sum(y_i - t_i)^2$$

2. Compute gradient

$$\nabla L = \frac{dL}{dw} = \frac{dL}{dh}\frac{dh}{dw}$$

$$if \;\; h_j = \sum w_{ji} x_i + b_j \rightarrow \frac{dh_j}{dw_{ji}} = \sum x_i$$

No activation function

$$\frac{dL}{dh} = \sum(y_i - t_i)$$

Backpropagation (IV)

1. Compute Loss

$$L_2 = \frac{1}{2}\sum (y_i - t_i)^2$$

2. Compute gradient

$$\nabla L = \frac{dL}{dw} = \frac{dL}{dh}\frac{dh}{dw}$$

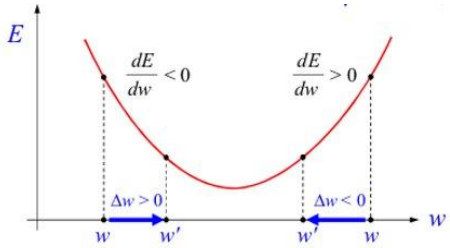
No activation function

$$if \;\; h_j = \sum w_{ji} x_i + b_j \rightarrow \frac{dh_j}{dw_{ji}} = \sum x_i$$

$$\frac{dL}{dh} = \sum (y_i - t_i) = \delta$$

3. Apply gradient descent

$$\Delta w = -\alpha \nabla L = -\alpha \delta \sum x_i$$



Backpropagation (V)

For output layer (L) with activation

$$\delta_j^L = \frac{dL}{dy}\frac{dy}{dh} = -(t_j - y_j^L) f'(h_j^L)$$

Backpropagation (VI)

For output layer (L) with activation

$$\delta_j^L = \frac{dL}{dy}\frac{dy}{dh} = -(t_j - y_j^L) f'(h_j^L)$$

But the same formula cannot be used for hidden neurons

$$\delta_i^{l-1} = f'(a_i^{l-1}) \sum (w_{ji}^l \delta_j^l)$$

Hence the idea is to propagate the errors back through the weights to assign a blame to hidden neuron

Backpropagation (VII)

For output layer (L) with activation

$$\delta_j^L = \frac{dL}{dy}\frac{dy}{dh} = -(t_j - y_j^L) f'(h_j^L)$$

But the same formula cannot be used for hidden neurons

$$\delta_i^{l-1} = f'(a_i^{l-1}) \sum (w_{ji}^l \delta_j^l)$$

Hence the idea is to propagate the errors back through the weights to assign a blame to hidden neuron

A single learning step requires:

- to show all the M examples;
- to store all the weights variations for each example;
- to update all the weights after completing the training set.

A pass of the entire training set is called an **epoch**

Backpropagation (VIII)

Initialize weights, η, ε, max_epochs; epoch = 0;
do {
  Initialize the global error: $E = 0$;
  for each $(x_k, t_k) \in$ TS { *epoch*
    compute y_{jk} $\forall$ layer (from 2 to L);
    compute δ_{jk} and $\Delta w_{ji}(k)$ $\forall$ layer (from L to 2);
    compute E_k and update global error: $E = E + E_k$;
  }
  update weights Δw_{ji} averaging on TS;
  epoch++;
} while (($E > \varepsilon$) AND (epoch < max_epochs));

For output layer (L) with activation

$$\delta_j^L = \frac{dL}{dy}\frac{dy}{dh} = -(t_j - y_j^L) f'(h_j^L)$$

But the same formula cannot be used for hidden neurons

$$\delta_i^{l-1} = f'(a_i^{l-1}) \sum (w_{ji}^l \delta_j^l)$$

Hence the idea is to propagate the errors back through the weights to assign a blame to hidden neuron

A single learning step requires:

- to show all the M examples;
- to store all the weights variations for each example;
- to update all the weights after completing the training set.

A pass of the entire training set is called an **epoch**

Limits of Gradient Descent (I)

For large networks or large training sets

- the backpropagation algorithm requires too much memory;

$$N = \prod_{l=1}^{L} (n_l + 1) n_{l+1} \rightarrow 8byte \ \ each$$

If L = 5 and n = 100, we need about 1 GB of RAM!

Limits of Gradient Descent (II)

For large networks or large training sets

- the backpropagation algorithm requires too much memory;

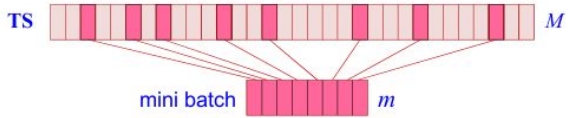
$$N = \prod_{l=1}^{L} (n_l + 1) n_{l+1} \rightarrow 8byte \ \ each$$

If L = 5 and n = 100, we need about 1 GB of RAM!

- since a learning step is performed after showing the entire training set, training can be too slow.

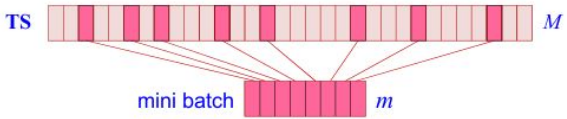
Stochastic Gradient Descent (I)

The Stochastic gradient descent (SGD) approach saves memory and time by estimating the gradient for a small number m of randomly chosen training inputs.



Stochastic Gradient Descent (II)

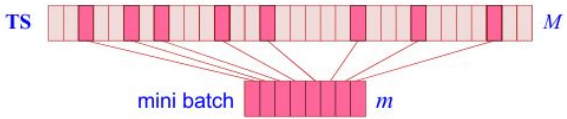
The Stochastic gradient descent (SGD) approach saves memory and time by estimating the gradient for a small number m of randomly chosen training inputs.



Initialize weights, η, ε, m, max_iter; iter $= 0$;
do {
  Initialize the global error: $E = 0$;
  Select a mini-batch of m random examples;
  for each $(x_k, t_k) \in$ mini-batch { *iteration*
    compute y_{jk} $\forall$ layer (from 2 to L);
    compute δ_{jk} and $\Delta w_{ji}(k)$ $\forall$ layer (from L to 2);
    compute E_k and update global error: $E = E + E_k$;
  }
  update all the weights averaging on mini-batch;
  iter++;
} **while** (($E > \varepsilon$) and (iter < max_iter));

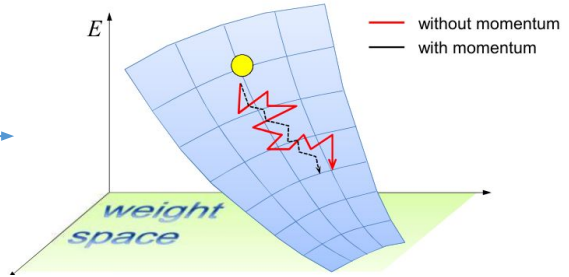
Stochastic Gradient Descent (III)

The Stochastic gradient descent (SGD) approach saves memory and time by estimating the gradient for a small number m of randomly chosen training inputs.



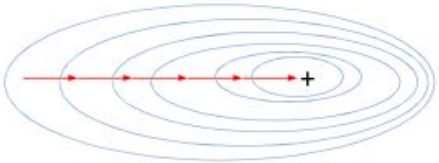
Initialize weights, η, ε, m, max_iter; iter = 0;
do {
  Initialize the global error: $E = 0$;
  Select a mini-batch of m random examples;
  for each $(x_k, t_k) \in$ mini-batch { *iteration*
    compute y_{jk} $\forall$ layer (from 2 to L);
    compute δ_{jk} and $\Delta w_{ji}(k)$ $\forall$ layer (from L to 2);
    compute E_k and update global error: $E = E + E_k$;
  }
  update all the weights averaging on mini-batch;
  iter++;
} **while** $((E > \varepsilon)$ and (iter < max_iter));

With Nesterov momentum

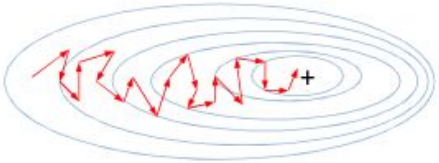


Stochastic Gradient Descent (IV)

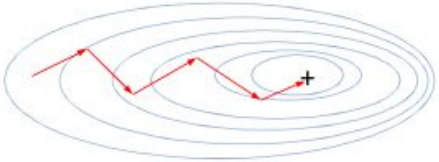
Batch mode: the gradient is computed on all the examples of the training set. Precise, but slow and memory consuming.



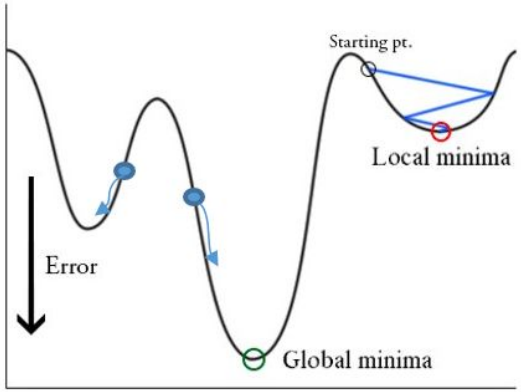
Stochastic Gradient Descent (**SGD**): the gradient is computed after every example. Much faster, but very noisy.



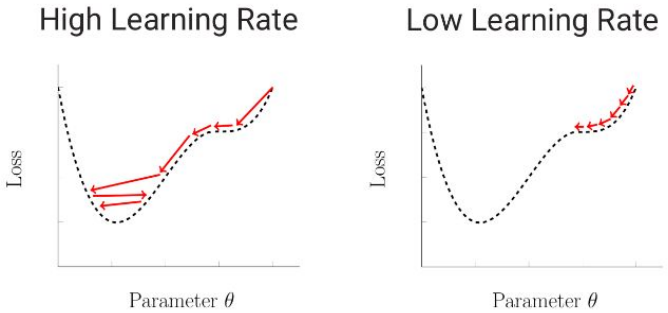
Mini batch mode: the gradient is computed on a random subset of examples (mini batch). It provides a balanced solution.



Importance of Hyperprameters (I)



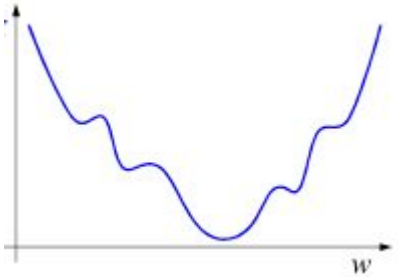
Importance of weight initialization

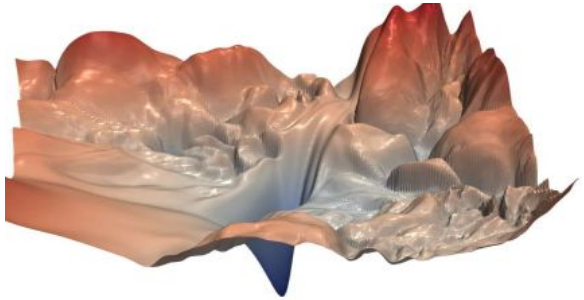


Importance of learning rate

Importance of Hyperprameters: Local minima (II)

Introducing non linearity (activation function)



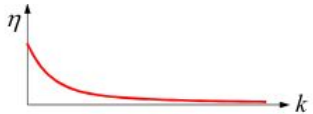


How solve the local minima problem?

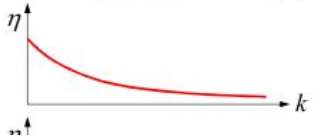
Importance of Hyperprameters: How speed learning (III)

If lambda is the learning rate decay and k is the epoch number, the following types of decay rules are commonly implemented:

Exponential decay $\eta = \eta_0 e^{-\lambda k}$



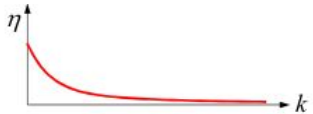
Time-based decay $\eta = \eta_0 \frac{1}{1 + \lambda k}$



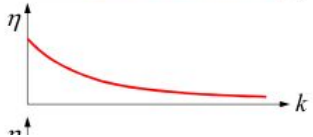
Importance of Hyperprameters: How speed learning (IV)

If lambda is the learning rate decay and k is the epoch number, the following types of decay rules are commonly implemented:

Exponential decay $\eta = \eta_0 e^{-\lambda k}$



Time-based decay $\eta = \eta_0 \frac{1}{1 + \lambda k}$



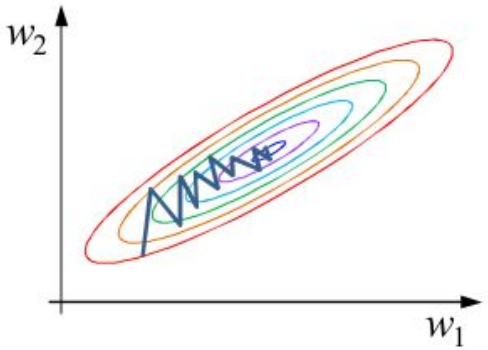
The momentum on mini-batch is a sort of low pass filter on weights variations

$$\Delta w_{i+1} = \mu \Delta w_i - \eta \nabla L$$

In this way, the next weight variation has 2 components: one is a fraction mu of the previous weight variation and another one that is a fraction eta of the error gradient in the previous step

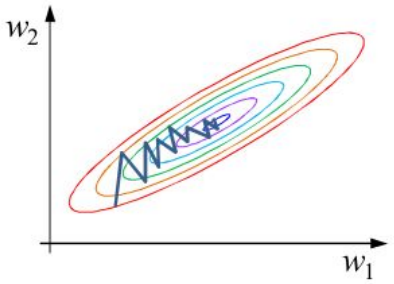
Importance of Hyperprameters: How speed learning (IV)

When data are not normalized, the Error function squeezes and a small variations can cause the error to oscillate



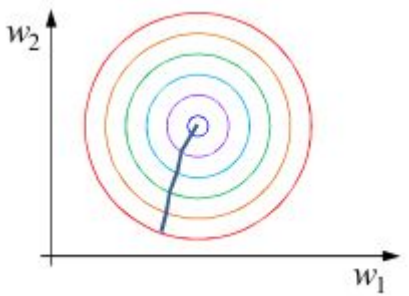
Importance of Hyperprameters: How speed learning (V)

When data are not normalized, the Error function squeezes and a small variations can cause the error to oscillate



With normalized data, the Error function is symmetric

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$



After normalization, input data have mean = 0 and std = 1

Measuring performance (I)

What it is really interesting in a neural network is its ability to generalize.

Measuring performance (II)

What it is really interesting in a neural network is its ability to generalize.

To evaluate the capability of generalizing the examples of the Training Set (TS), we define another set of examples, called the Validation Set (VS).

Measuring performance (III)

What it is really interesting in a neural network is its ability to generalize.

To evaluate the capability of generalizing the examples of the Training Set (TS), we define another set of examples, called the Validation Set (VS).

Size	percentage	number
Small data sets (100 examples)	70 % for TS 30 % for VS	70 for TS 30 for VS
Medium data sets (10,000 examples)	90 % for TS 10 % for VS	9,000 for TS 1,000 for VS
Large data sets (1,000,000 examples)	99 % for TS 1 % for VS	990,000 for TS 10,000 for VS

Measuring performance (IV)

What it is really interesting in a neural network is its ability to generalize.

To evaluate the capability of generalizing the examples of the Training Set (TS), we define another set of examples, called the Validation Set (VS).

Size	percentage	number
Small data sets (100 examples)	70 % for TS 30 % for VS	70 for TS 30 for VS
Medium data sets (10,000 examples)	90 % for TS 10 % for VS	9,000 for TS 1,000 for VS
Large data sets (1,000,000 examples)	99 % for TS 1 % for VS	990,000 for TS 10,000 for VS

We could use 2 principal score:

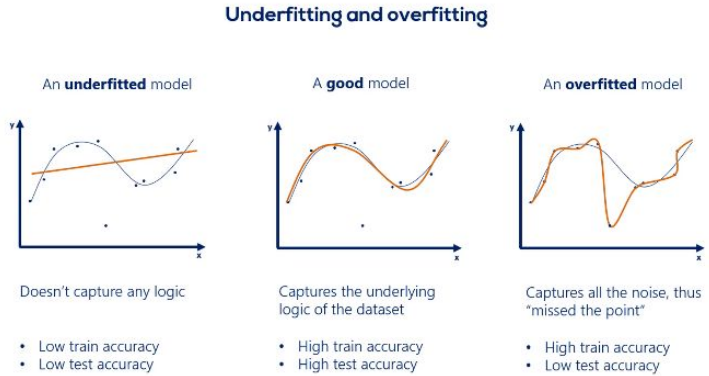
$$\text{Accuracy} = 100 * \frac{\textit{Number of correct outputs}}{\textit{Total number of test samples}}$$

		True labels: cat	True labels: dog
Predicted label	cat	**15**	4
	dog	5	**6**

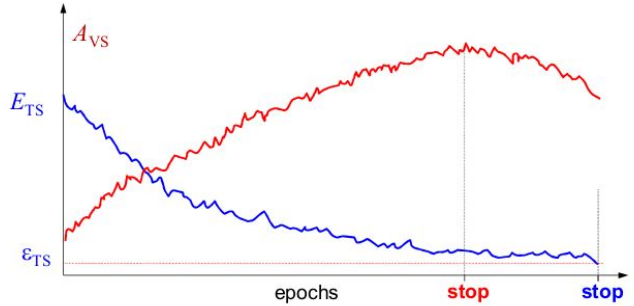
Confusion matrix (for classification)

Underfitting and Overfitting

Overfitting is the situation in which a neural network starts learning too much details of the TS, loosing the ability to generalize on new examples.



Train error	Test error		
1%	11%	High variance	Overfitting
15%	16%	High bias	Underfitting
1,5%	3%	Low bias/variance	Just right
15%	30%	High bias/variance	WRONG



Observing accuracy during training we can show the overfitting and we could stop training earlier.

How solve overfitting (I)

We have 2 choice:

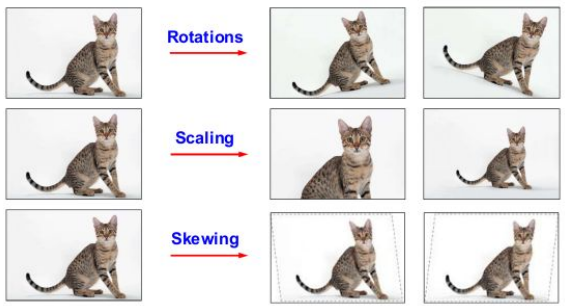
- More data
- Regularization

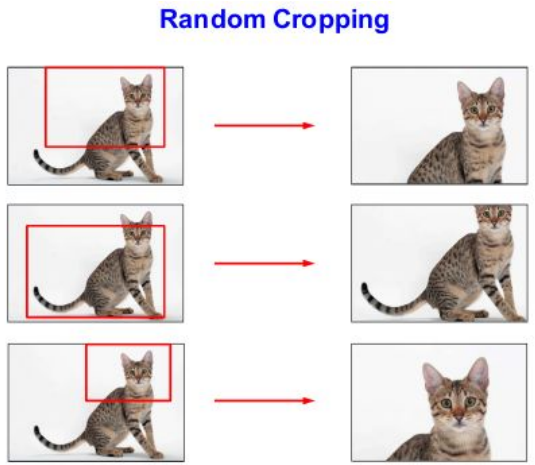
How solve overfitting (II)

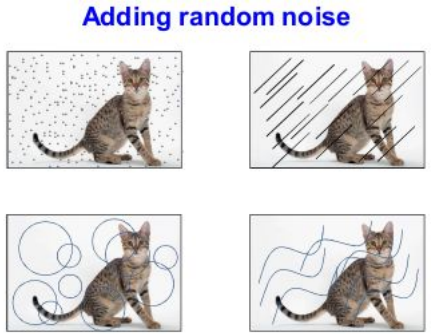
We have 2 choice:

- More data
- Regularization

Augumentation







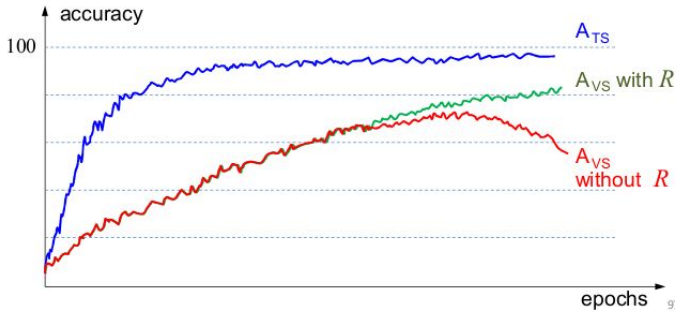
How solve overfitting (III)

We have 2 choice:

- More data
- Regularization

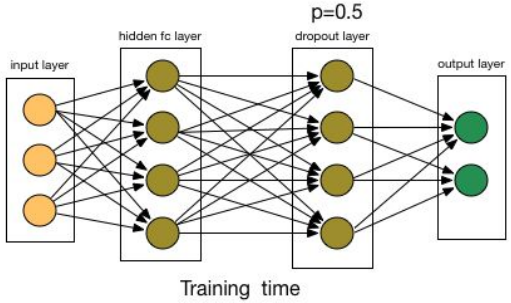
The idea is to add an extra regularization term R to the cost function

$$L = L + \frac{\lambda}{2M} \sum w^2$$



Dropout and batch normalization

- **Dropout**



Another method to prevent overfitting

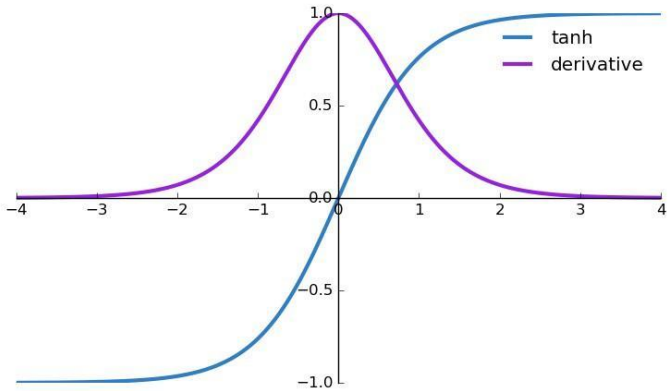
- We have seen that input normalization helps learning, since it makes the Loss function more symmetric. As input values pass through many layers, activations lose normalization and this can slow down learning. To compensate for that, **Batch Normalization** consists in normalizing the activations also in the hidden layers. Batch normalization is usually done before non linearity

To reduce the vanishing of gradient

Vanishing and exploding of gradient

increasing the number of hidden layers leads to two problems:

1. **Vanishing gradient**: as we add more and more hidden layers, Backpropagation becomes less and less effective to the lower layers, since the gradient becomes smaller and smaller (hence vanishing).
2. **Exploding gradient**



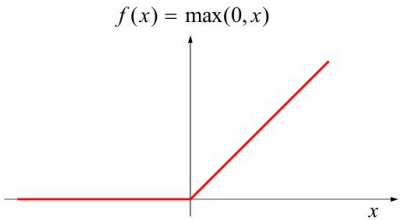
During training the weights grow, so the output of derivative of activation function get a small value. During backpropagation delta tend to vanish, due to chain products.

$$\delta_i^{l-1} = f'(a_i^{l-1}) \sum (w_{ji}^{l} \delta_j^{l})$$

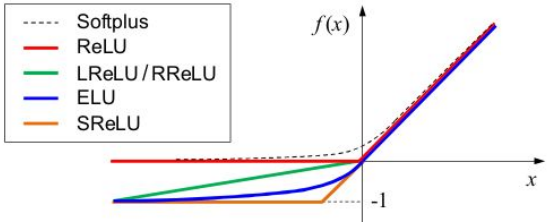
A good solution could be initialize the weights with small number (and batch normalization)

Influence of activation function on vanishing gradients

In both sigmoid and tanh, neurons stop learning when they saturate. A common solution is to adopt a neuron with a rectified linear activation function, also called Rectified Linear Unit (ReLU).



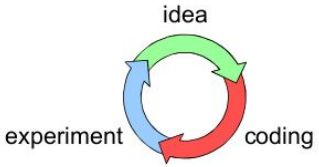
Increasing weights and the input activation, a rectified linear unit will never saturate, so there is no learning slowdown. On the other hand, when the activation is negative, the gradient vanishes, so the neuron stops learning.

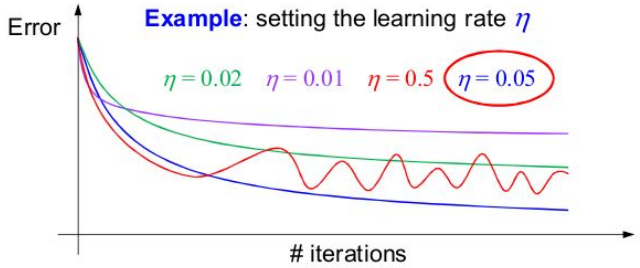


How set hyperparameters

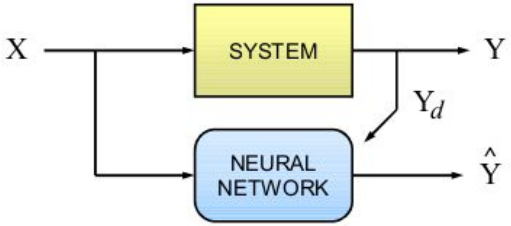
There are a lot of hyper parameters:

- Hidden layers
- Number of neurons for layer
- Learning rate
- Momentum
- Weight initialization
- Batch size
- Epochs
- Activation functions
- Dropout probability
- Regularization constant
- ...

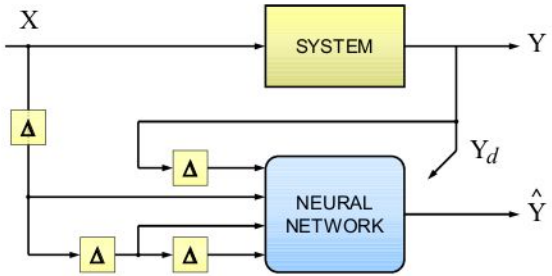




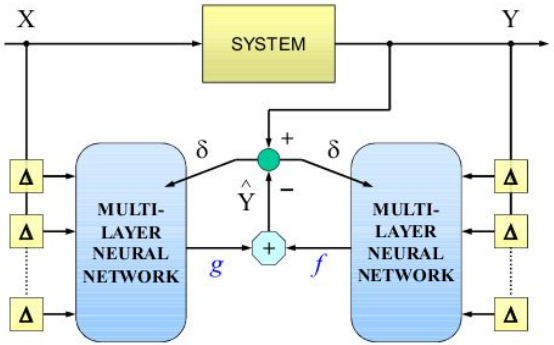
Other applications of deep learning



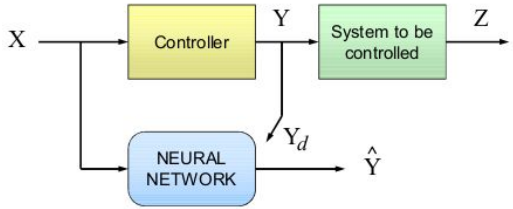
System identification



LTI



Non linear system



Copying behavior of a controller

Deep learning: What's next?

In this course

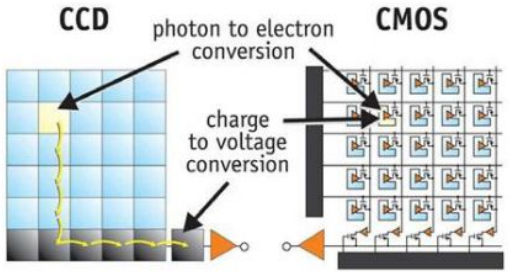
- Computer vision
- Convolutional Neural Networks

- Recurrent Neural Networks
- Reinforcement learning

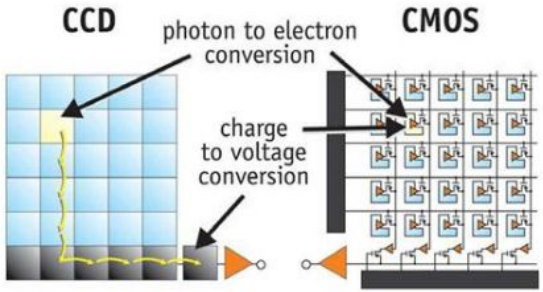
Agenda : Lesson 3

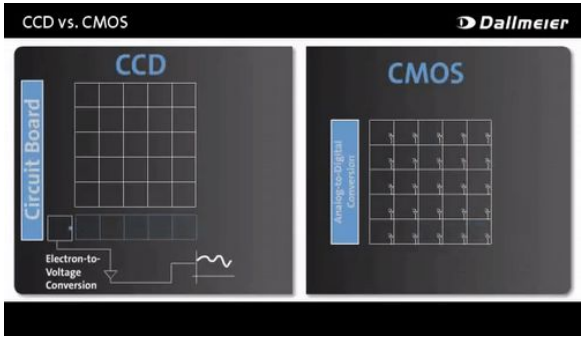
- Artificial vision
- Classical CV
  - How use filtering for image manipulation
- Deep neural network for computer vision
  - Basic theory
  - State of the art architecture
  - Transfer learning

Computer vision: Sensors (I)

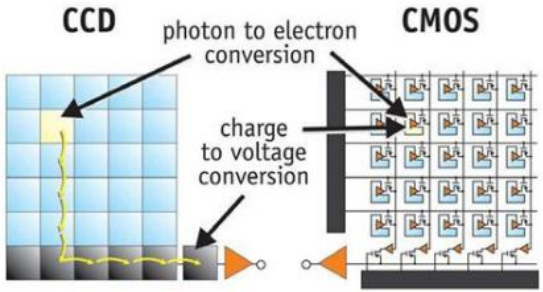


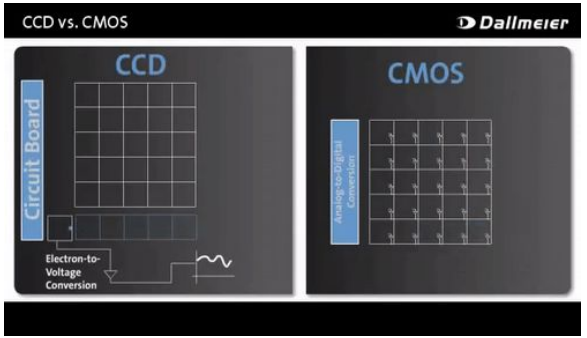
Computer vision: Sensors (II)



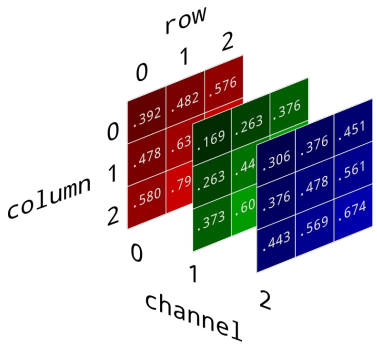


Computer vision: Sensors (III)



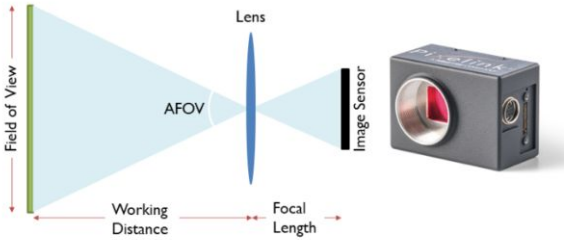


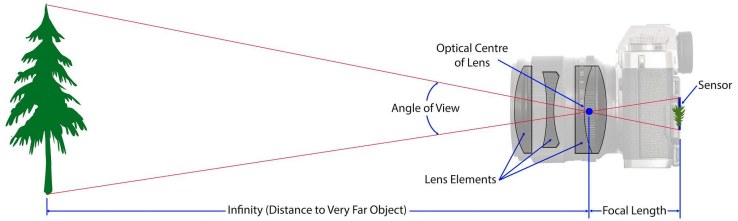
Digital Conversion



In range of 0-255 for each channel if uint(8) representation of the data.

Computer vision: Sensors (IV)





- **Sensitivity**: Measure of minimum lux
- **SNR** : Industrial camera has SNR > 50dB
- **Frame rate**: image for second
- **Fill factor**
- **Resolution**
- **Sensor size**

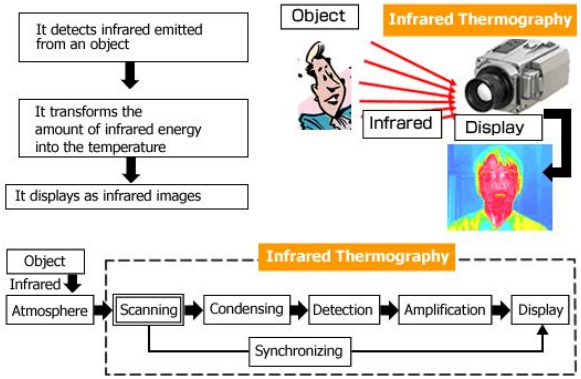
$$HFOV = w \cdot AFOV = w \cdot 2atan^{-1}\frac{WD}{2f}$$

$$VFOV = HFOV * \frac{h}{w}$$

Sensors with highest fill factor and size have better performance (highest SNR and quantity of lux).

CCD sensors have better performance.

Computer vision: Sensors (IV)



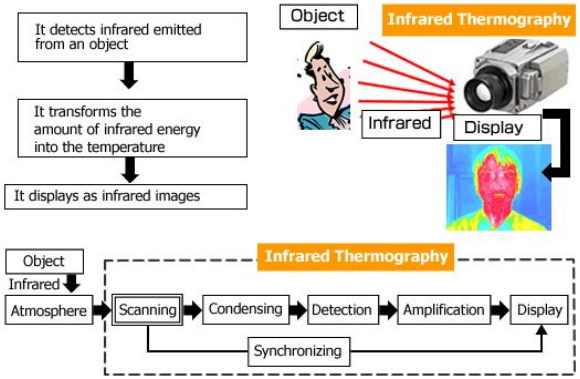
The thermal image shows the temperature distribution on the surface of an object.
They are divided into

- Long wave: with wavelengths of about 3 - 5μm for the analysis of bodies at high temperature
- Short wave with wavelengths of 8 - 14μm for measurements of temperatures close to ambient temperatures

The data collected by the camera depends on the emissivity of the body. The lower the emissivity:

- the higher the share of reflected infrared rays
- the more difficult it is to measure the temperature accurately.

Computer vision: Sensors (V)



The thermal image shows the temperature distribution on the surface of an object.
They are divided into

- Long wave: with wavelengths of about 3 - 5μm for the analysis of bodies at high temperature
- Short wave with wavelengths of 8 - 14μm for measurements of temperatures close to ambient temperatures

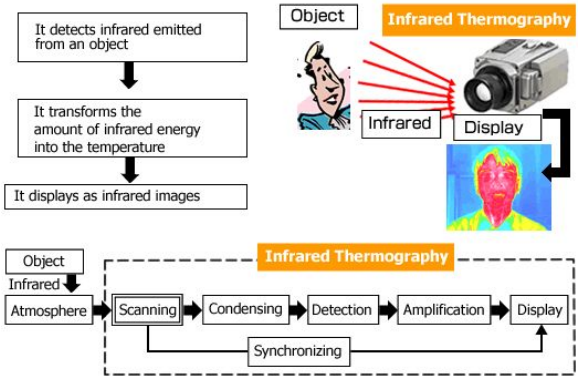
The data collected by the camera depends on the emissivity of the body. The lower the emissivity:

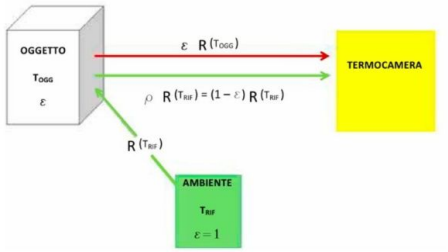
- the higher the share of reflected infrared rays
- the more difficult it is to measure the temperature accurately.

The characteristics of a thermal camera are like those of a standard camera.

- In a microbolometer sensor the value of the frame rate is limited due to the inertia of the detectors (for uncooled sensors) to about 60 Hz.
- Cameras with an acquisition frequency higher than 60 Hz cover most applications, for which it is necessary to choose cameras with quantum, therefore cooled and more expensive

Computer vision: Sensors (VI)





We must calibrate it.

The thermal image shows the temperature distribution on the surface of an object.
They are divided into

- Long wave: with wavelengths of about 3 - 5μm for the analysis of bodies at high temperature
- Short wave with wavelengths of 8 - 14μm for measurements of temperatures close to ambient temperatures

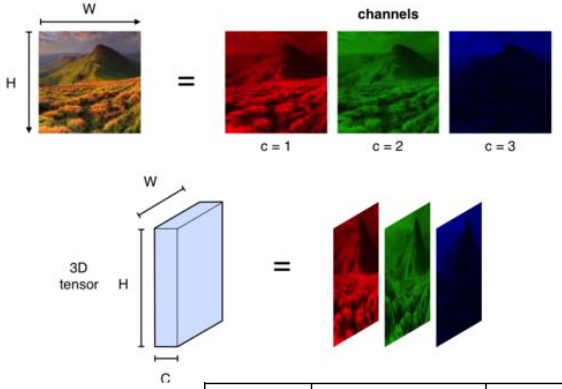
The data collected by the camera depends on the emissivity of the body. The lower the emissivity:

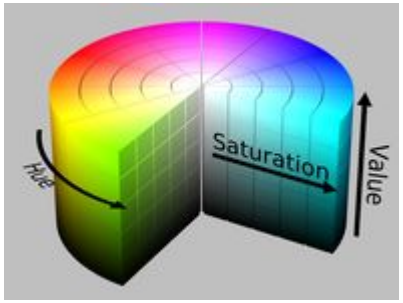
- the higher the share of reflected infrared rays
- the more difficult it is to measure the temperature accurately.

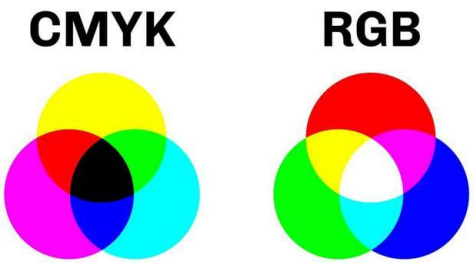
The characteristics of a thermal camera are like those of a standard camera.

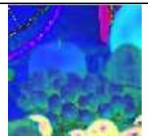
- In a microbolometer sensor the value of the frame rate is limited due to the inertia of the detectors (for uncooled sensors) to about 60 Hz.
- Cameras with an acquisition frequency higher than 60 Hz cover most applications, for which it is necessary to choose cameras with quantum, therefore cooled and more expensive

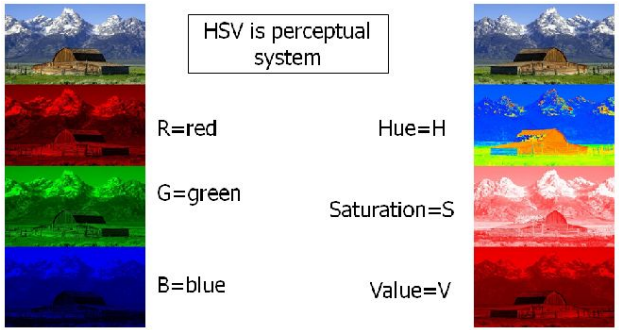
Computer vision: Color space



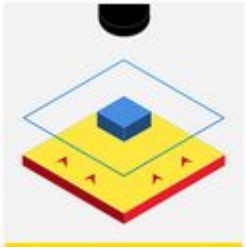




Input image	Original image	RGB → HSV
girl		
fruits		



Computer vision: Illumination systems (I)

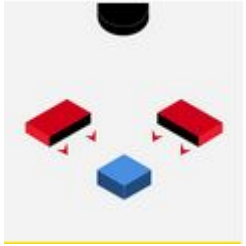


Back lighting projects even illumination from behind the target highlighting the silhouette of the target. This lighting type is used to detect the presence/absence of holes or gaps, measurement or verification of the target outline shape, as well as enhancing cracks, bubbles, and scratches on clear target parts. Note that surface detail is lost with this lighting type.

The **In-Sight integrated light** is a diffuse ring light that provides bright uniform lighting on the target for machine vision applications. This integrated light reduces shadowing and provides uniform illumination on matte objects. Due to the diffuse nature of the light, at closer working distances (less than 70mm), the light also provides a dark field lighting technique.

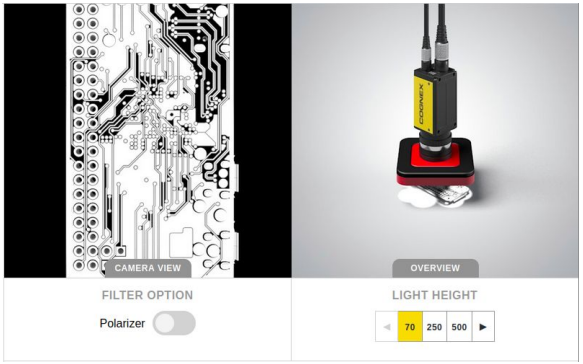
Computer vision: Illumination systems (II)

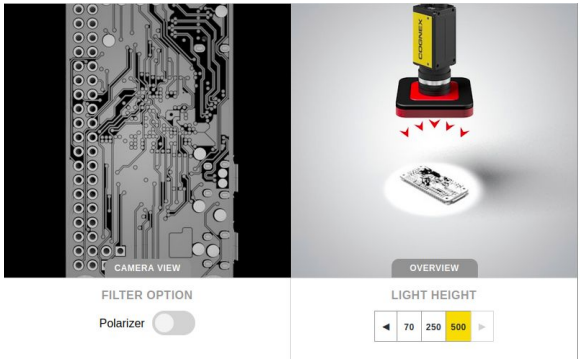
Ring lighting is a circle or ring of bright, intense lighting that provides shadow-free illumination and good image contrast. Ring lighting is a common lighting type covering a broad range of applications due to its versatility. Note that it can cause specular glare on reflective parts.

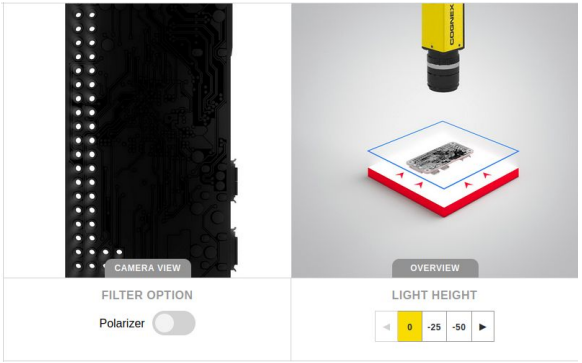


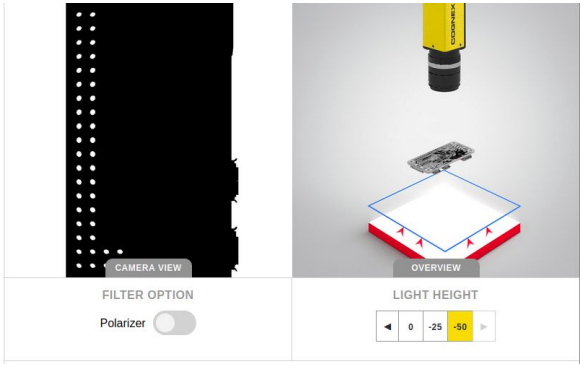
Dark field lighting technique provides a light at a shallow angle to the target. Any surface features–scratches, edges, imprints, notches–reflect the light back to the camera making these surface features appear bright, while the rest of the surface is dark. This technique can be created with any directional lighting option (bar, ring, spot) that allows for the light to be angled to the part's surface. This lighting type is most often used for surface inspections where strong contrast on edges is needed, such as glass inspection and reading laser-etched codes. It enhances the contrast of surface features such as laser embossed or engraved marks.

Computer vision: Illumination systems (III)



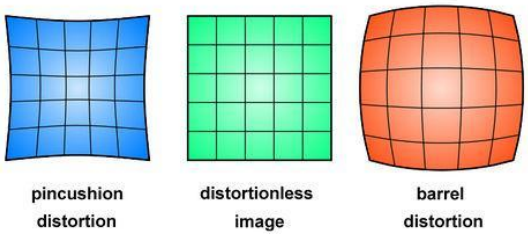


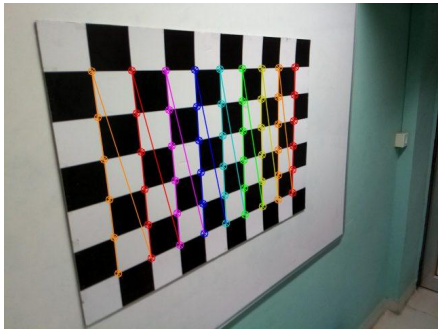




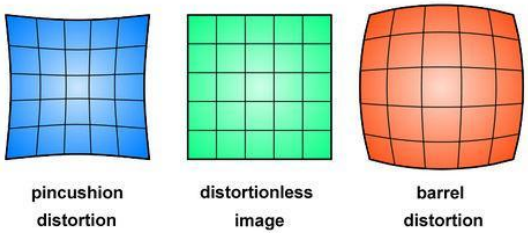
https://www.cognex.com/resources/interactive-tools/lighting-advisor

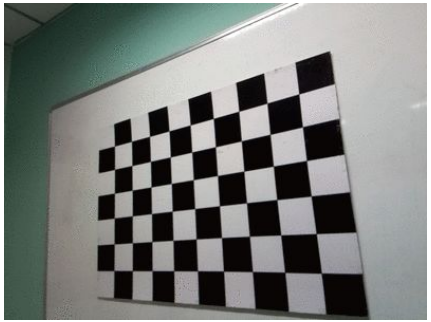
Computer vision: Image distortions and camera calibration (I)

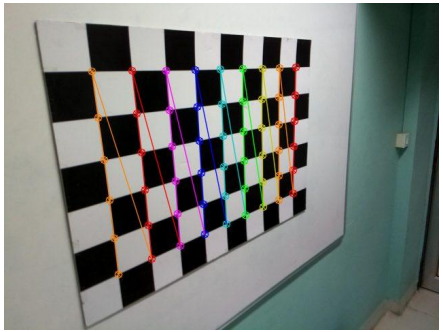


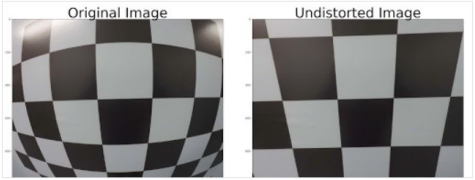


Computer vision: Image distortions and camera calibration (II)

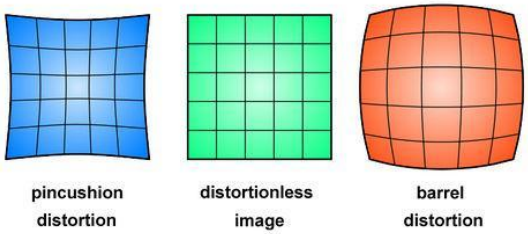


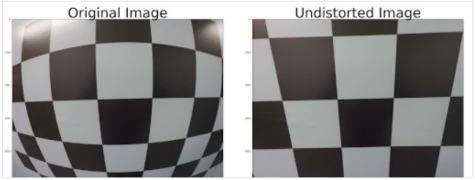






Computer vision: Image distortions and camera calibration (III)





Camera Calibration Flowchart

Define real world coordinates of 3D points using checkerboard pattern of known size.

↓

Capture the images of the checkerboard from different viewpoints.

↓

Use **findChessboardCorners** method in OpenCV to find the pixel coordinates (u, v) for each 3D point in different images

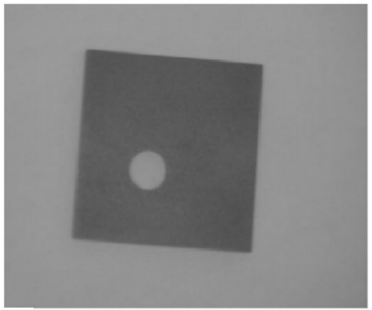
↓

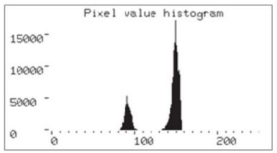
Find camera parameters using **calibrateCamera** method in OpenCV, the 3D points, and the pixel coordinates.

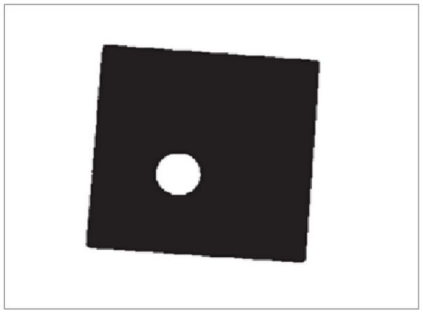
Computer vision: Histogram, grayscale and binarization

There are some cameras that does not have the capacity to capture color (**monocromatic).** The digital result is a matrix (2D tensor) that contain only information about **gray level.**

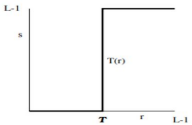
- From any image we can get the greyscale one with different methods.
- From a grayscale image we can get the binary one (b/w) observing the histogram of pixel's intensity.
- Founded a threshold in this way we can apply a low pass filter on image:



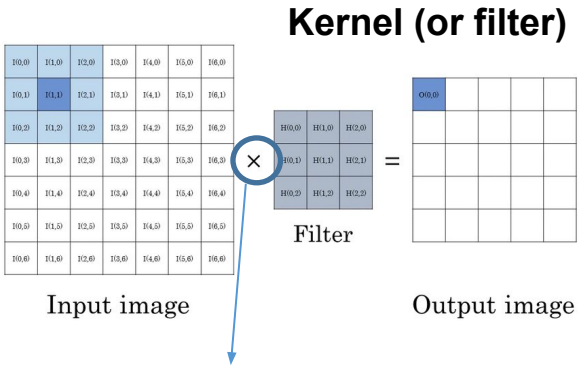


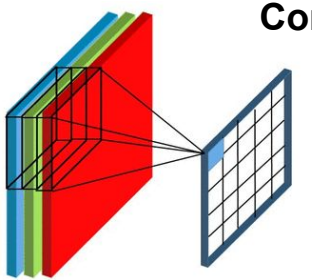


$$s = \{0\ L-1\ \frac{if\ r \le T}{if\ r > T}$$

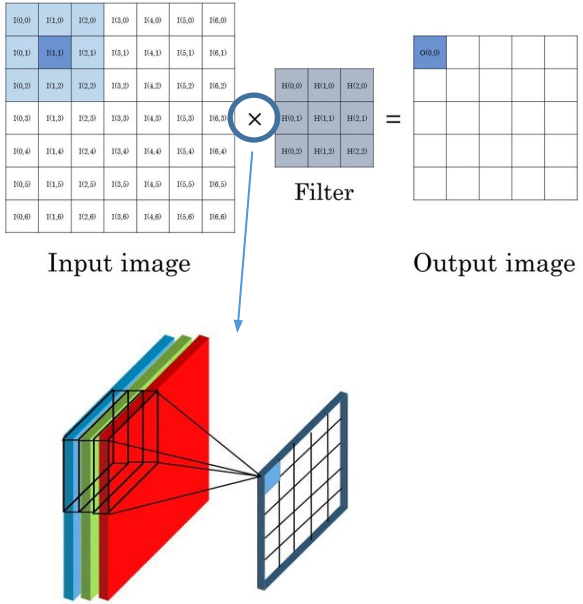


Computer vision: Convolution operation and filters (I)

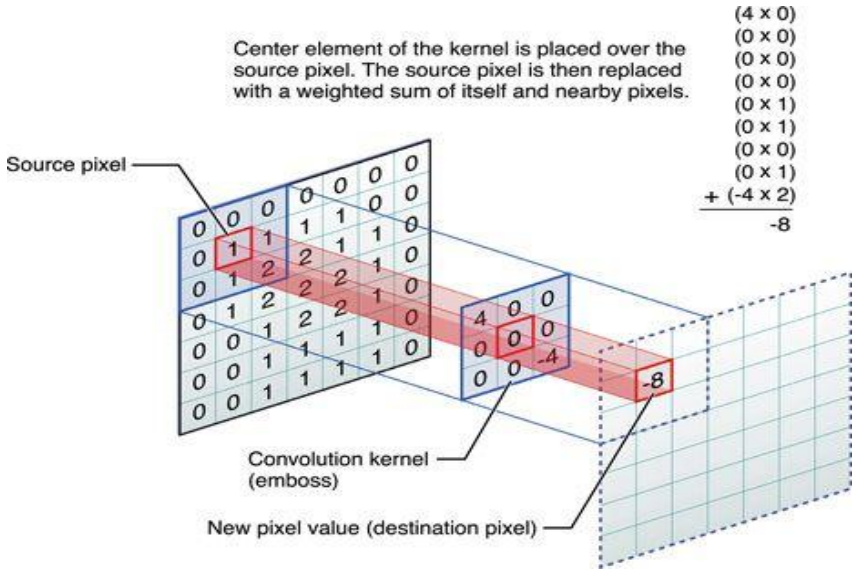




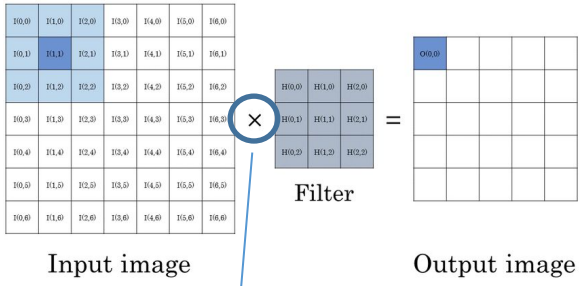
Computer vision: Convolution operation and filters (II)



$$\boldsymbol{O}[u,v] = \sum_{i=-h}^{h} \sum_{j=-h}^{h} \boldsymbol{I}[u-i, v-j]\boldsymbol{K}[i,j] \ , \qquad \forall (u,v) \in \boldsymbol{I}$$



Computer vision: Convolution operation and filters (III)

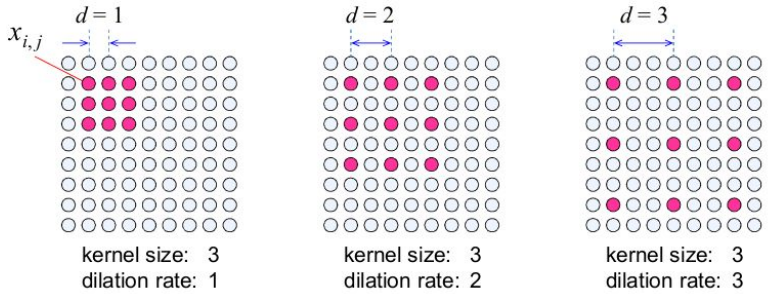


The output image has a new size:

$$o = d - f + 1$$

But I can decide to use a dilatation rate or slide different from 1:

$$o = \frac{d - f}{s} - 1$$



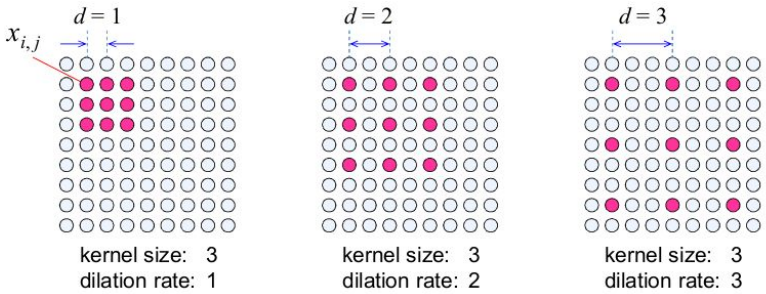
Computer vision: Convolution operation and filters (IV)

The output image has a new size:

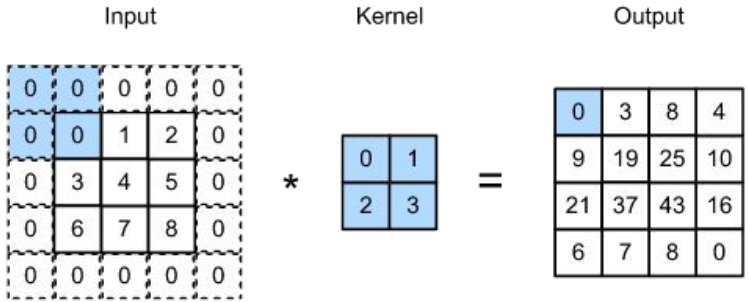
$$o = d - f + 1$$

But I can decide to use a dilatation rate or slide different from 1:

$$o = \frac{d - f}{s} + 1$$



In this way pixels on the border are less affected by the convolution, hence features on the border could not be detected. To solve this problme we can introduce padding: **adding an extra border** filled by zeros.



$$o = \frac{d - f + 2p}{s} + 1$$

Computer vision: Convolution operation and filters (IV)

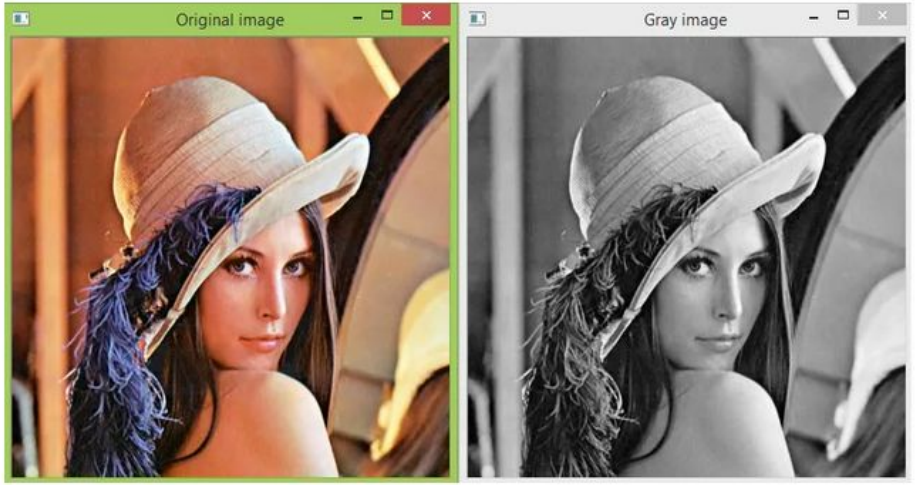
In this way pixels on the border are less affected by the convolution, hence features on the border could not be detected. To solve this problme we can introduce padding: **adding an extra border** filled by zeros.

$$o = \frac{d - f + 2p}{s} + 1$$

We can distinguish 2 different kind of padding:

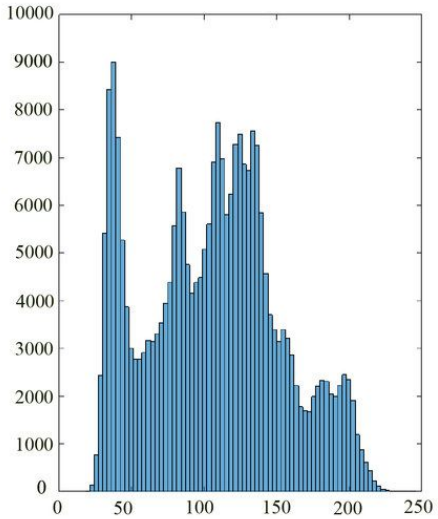
- Same, to have the same size for output image $p = \frac{s * (o - 1) - n + f}{2}$
- Valid – no padding

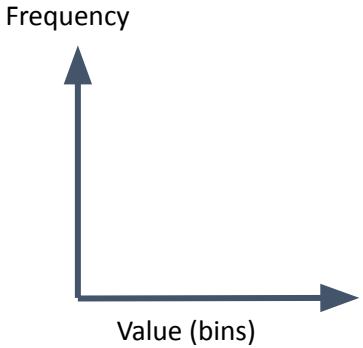
Coding: Gray scale image




```
import cv2

image = cv2.imread('C:/Users/N/Desktop/Test.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Original image',image)
cv2.imshow('Gray image', gray)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

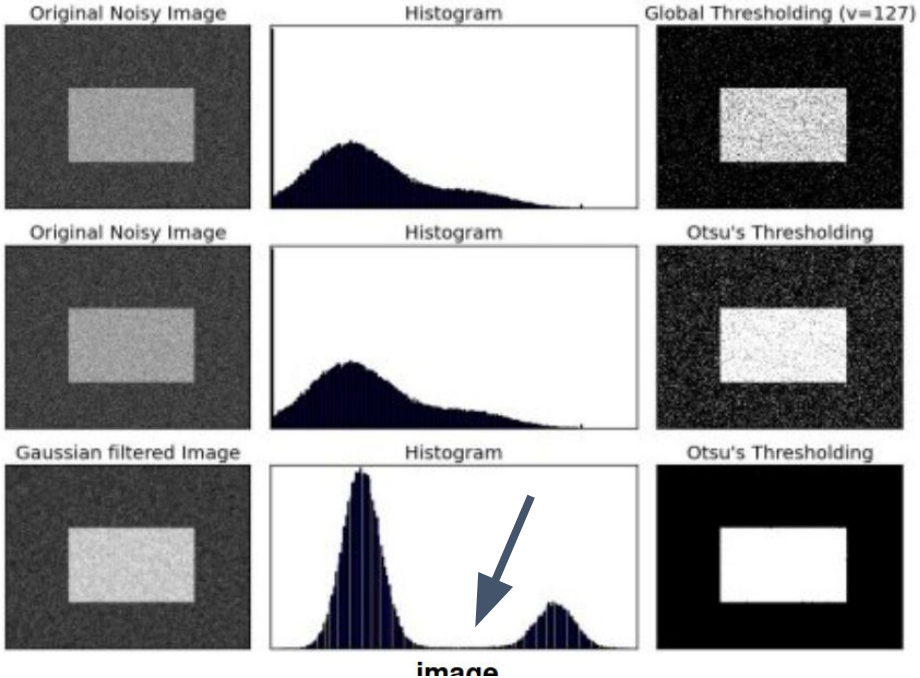
Coding: Histogram

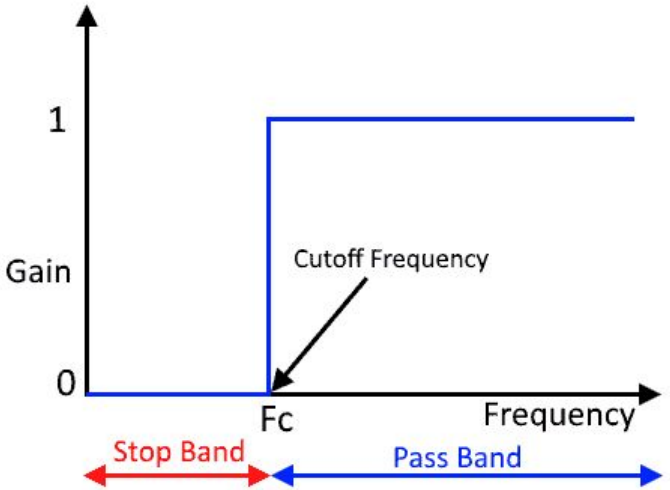




https://www.khanacademy.org/math/statistics-probability/displaying-describing-data/quantitative-data-graphs/a/histograms-review

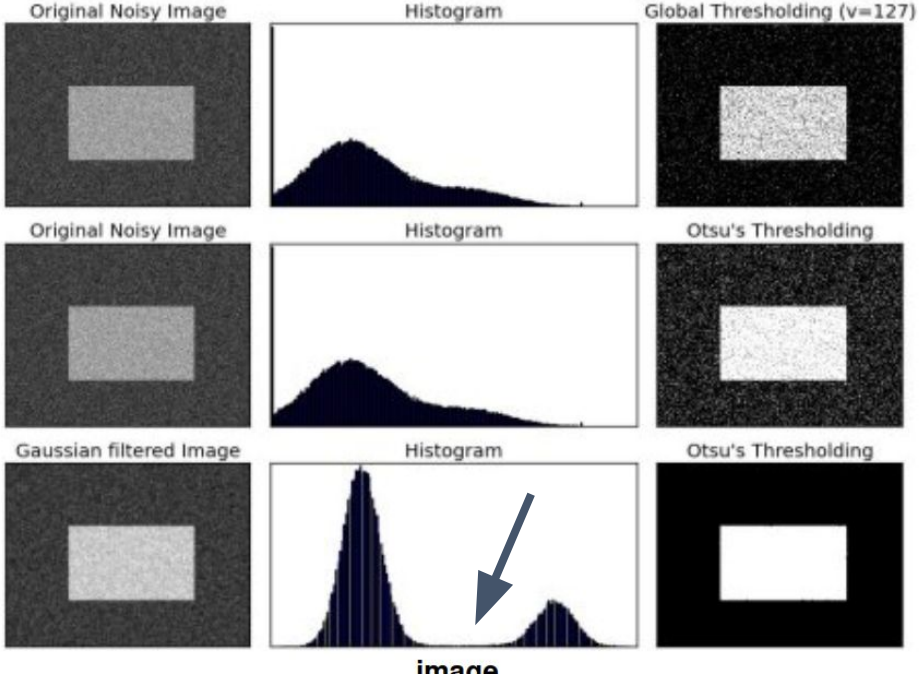
Coding: Binarization





https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Coding: Binarization



image

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('noisy2.png',0)
# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Coding: Image gradients (Sobel)

The Sobel Operator is a discrete differentiation operator. It computes an approximation of the gradient of an image intensity function.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

TOTAL GRADIENT (feature)

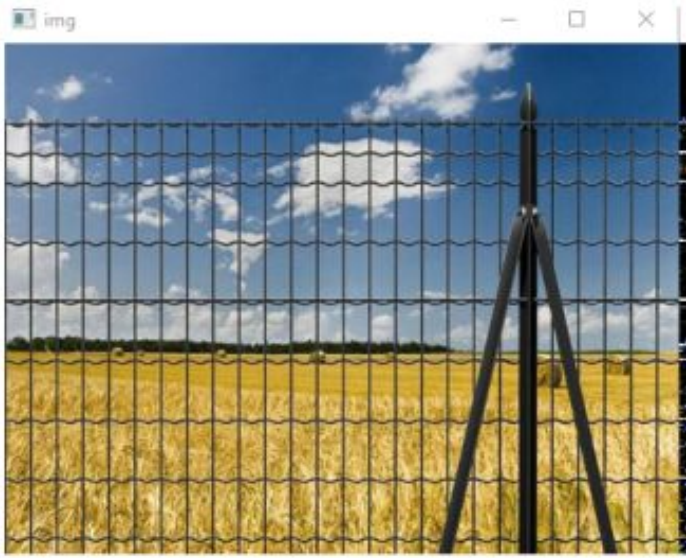
$$G = \sqrt{G_x^2 + G_y^2}$$

OR

$$G = |G_x| + |G_y|$$

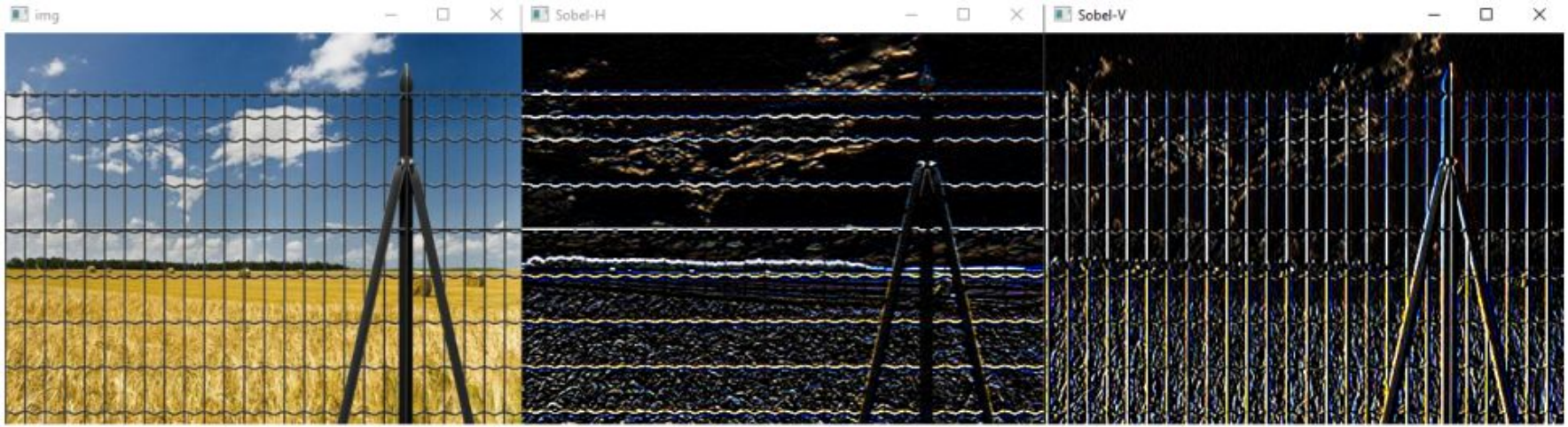
https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Coding: Image gradients (Sobel)




```python
"""
Esercitazione python su filtri e convoluzioni
DATA: 21/05/2021
PROPRIETA: Mario Vozza
"""
import cv2
import numpy as np

#lettura immagine NB: Posizionarsi nella cartella dove è posizionata l'immagine

img = cv2.imread('recinzione.jpg')
img = cv2.resize(img,(450,338))
cv2.imshow('img', img)

#filtro di Sobel orizzontale
Sobel = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
Sobeled = cv2.filter2D(img, -1, Sobel)
cv2.imshow('Sobel-H', Sobeled)

#filtro di Sobel verticale
SobelV = Sobel.T
SobeledV = cv2.filter2D(img, -1, SobelV)
cv2.imshow('Sobel-V', SobeledV)

#positioning
cv2.moveWindow('img', 0, 0)
cv2.moveWindow('Sobel-H', 450, 380)
cv2.moveWindow('Sobel-V', 900, 380)

# attesa e chiusura schede
cv2.waitKey()
cv2.destroyAllWindows()
```

Coding: Image gradients (Sobel)



Coding : Average Blur

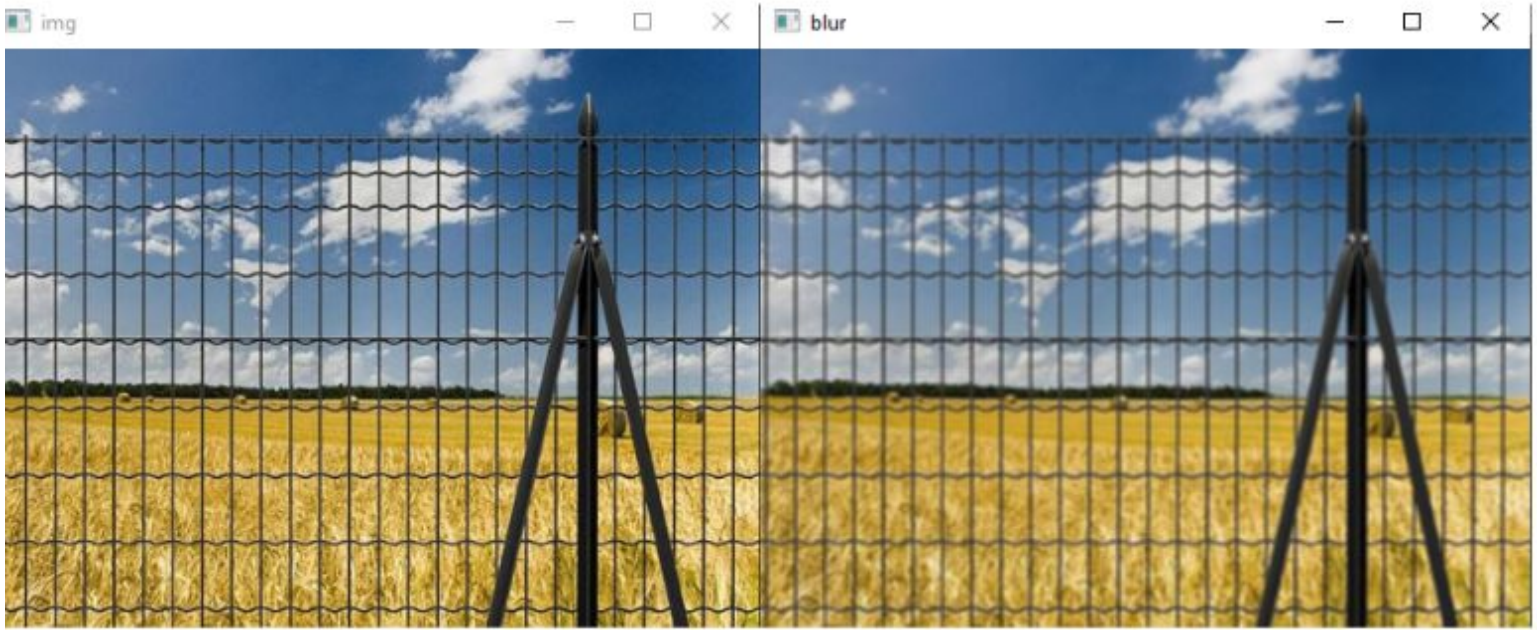
As in one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. LPF helps in removing noise, blurring images, etc. HPF filters help in finding edges in images. OpenCV provides a function **cv.filter2D()** to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel will look like the below:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('opencv_logo.png')

kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)
```

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Keep this kernel above a pixel, add all the 25 pixels below this kernel, take the average, and replace the central pixel with the new average value.

Coding : Average Blur



Coding : Gaussian Blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

3 x 3 Gaussian Kernel

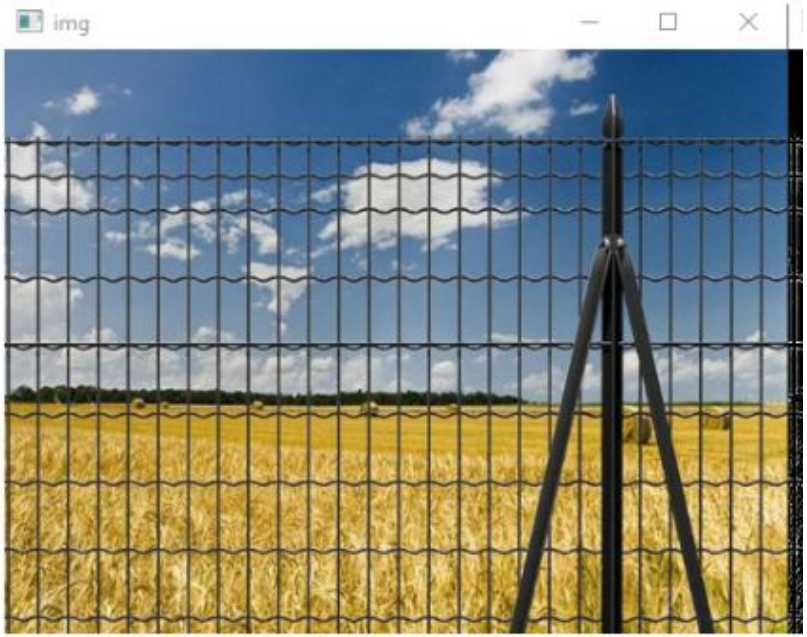
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

5 x 5 Gaussian Kernel

The standard deviation of the Gaussian distribution in X and Y direction should be chosen carefully considering the size of the kernel such that the edges of the kernel is close to zero.You have to choose a right size of the kernel to define the neighborhood of each pixel. If it is too large, small features of the image may be disappeared and the image will look blurred. If it is too small, you cannot eliminate noises in the image.

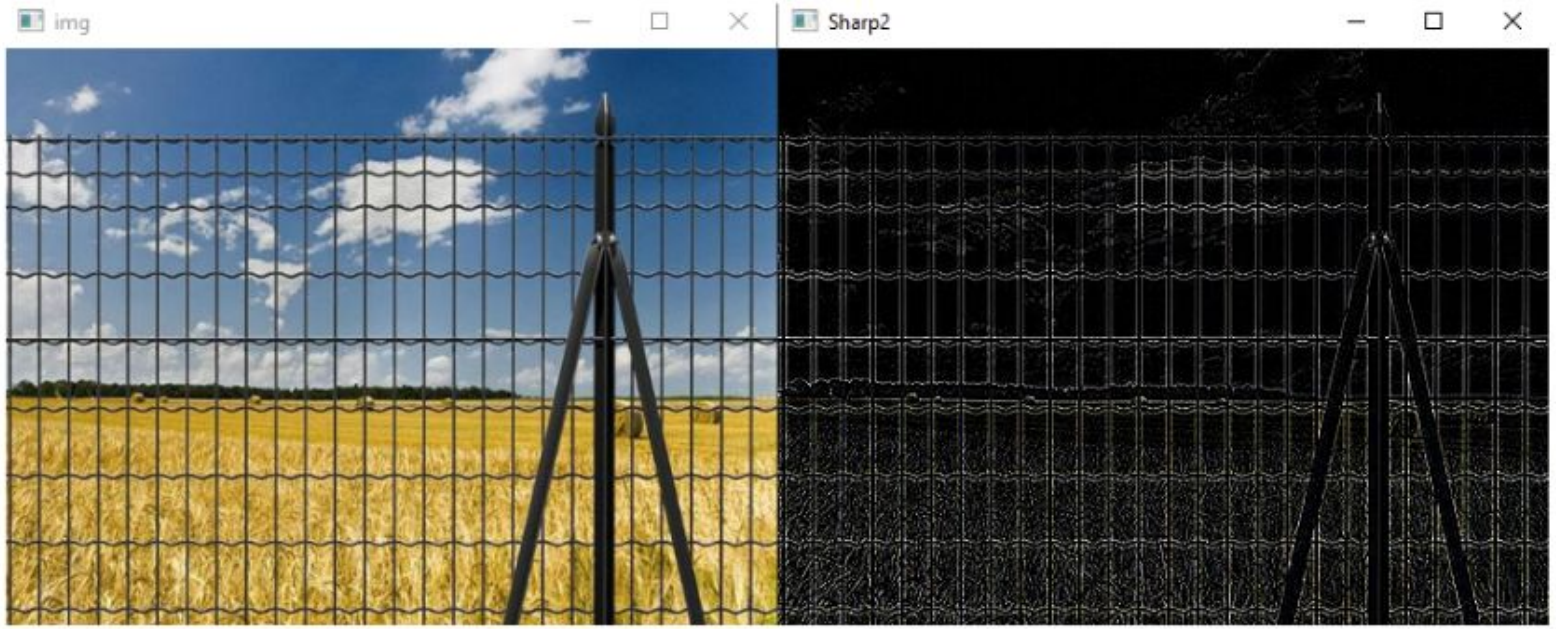
```
std = 3
blur = cv.GaussianBlur(img,(5,5), std)
```

Coding: Sharpening filter 0 mean




```python
# -*- coding: utf-8 -*-
"""
Esercitazione python su filtri e convoluzioni
DATA: 21/05/2021
PROPRIETA: Mario Vozza
"""
import cv2
import numpy as np

#lettura immagine NB: Posizionarsi nella cartella dove è posizionata l'immagine

img = cv2.imread('recinzione.jpg')
img = cv2.resize(img,(450,338))
cv2.imshow('img', img)

#filtro di Sharpen nullo
Sharp2 = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
Sharpen2 = cv2.filter2D(img, -1, Sharp2)
cv2.imshow('Sharp2', Sharpen2)

#positioning
cv2.moveWindow('img', 0, 0)
cv2.moveWindow('Sharp2', 0, 380)

# attesa e chiusura schede
cv2.waitKey()
cv2.destroyAllWindows()
```

Coding: Sharpening filter 0 mean



Coding: Morphological filter

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Morphological filter

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Morphological filter

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

Erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Morphological filter

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Morphological filter

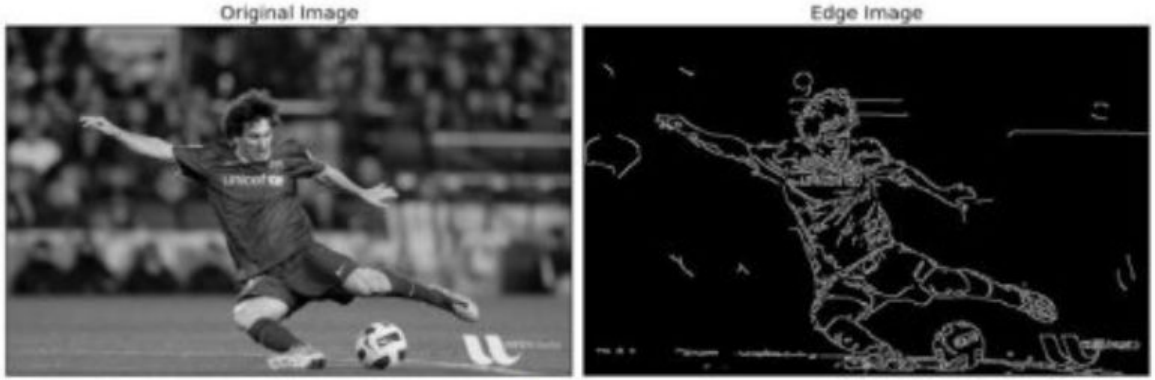
```python
import cv2 as cv
import numpy as np
img = cv.imread('j.png',0)
kernel = np.ones((5,5),np.uint8) # square kernel

erosion = cv.erode(img,kernel,iterations = 1)

dilation = cv.dilate(img,kernel,iterations = 1)

opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)

closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

```
# Rectangular Kernel
>>> cv.getStructuringElement(cv.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```


https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Canny edge detector

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

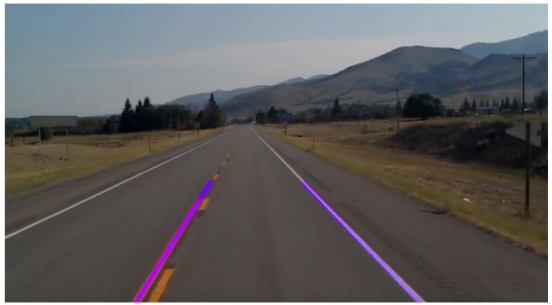
Coding: Hough transform

The Hough transform can be used to identify the parameter(s) of a curve which best fits a set of given edge points. This edge description is commonly obtained from a feature detecting operator such as:

- Laplacian
- Sobel
- Canny
- Roberts Cross
- …

Works also for noisy features, *i.e.* it may contain multiple edge fragments corresponding to a single whole feature. Furthermore, as the output of an edge detector defines only *where* features are in an image, the work of the Hough transform is to determine both *what* the features are and *how many* of them exist in the image.

- Ex. used for line detection



https://blog.paperspace.com/understanding-hough-transform-lane-detection/

https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Coding: Hough transform

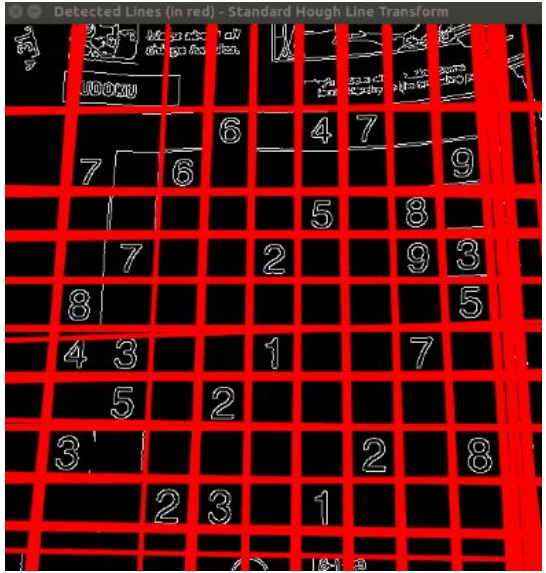
```
#  Standard Hough Line Transform
lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)
```

- with the following arguments:
  - *dst*: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
  - *lines*: A vector that will store the parameters (r, θ) of the detected lines
  - *rho* : The resolution of the parameter r in pixels. We use **1** pixel.
  - *theta*: The resolution of the parameter θ in radians. We use **1 degree** (CV_PI/180)
  - *threshold*: The minimum number of intersections to "*detect*" a line
  - *srn* and *stn*: Default parameters to zero. Check OpenCV reference for more info.

https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Coding: Hough transform

```python
import math
import cv2 as cv
import numpy as np

def main(src):

    dst = cv.Canny(src, 50, 200, None, 3)

    # Copy edges to the images that will display the results in BGR
    cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
    cdstP = np.copy(cdst)

    lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

    if lines is not None:
        for i in range(0, len(lines)):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            a = math.cos(theta)
            b = math.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
            pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
            cv.line(cdst, pt1, pt2, (0,0,255), 3, cv.LINE_AA)


    linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

    if linesP is not None:
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE_AA)

    cv.imshow("Source", src)
    cv.imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst)
    cv.imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP)
    cv.waitKey()
    return 0
```



https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Coding: Seam tracker offline

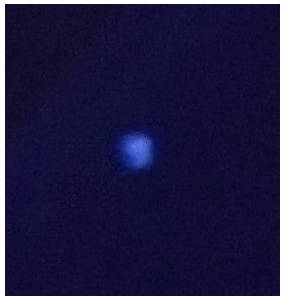
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

VIDEO = r'D:\Giulio\Condivisa PC pvt-az\Notos Ad Tech\RD_prj\Path following\color.mp4'
cap = cv2.VideoCapture(VIDEO)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))

while True:

    _,frame = cap.read()

    img2 = frame[:,:,2]
    _, img = cv2.threshold(img2, 160, 255, cv2.THRESH_BINARY)

    img = cv2.Canny(img, 20, 200, None, 3)
    img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    lines = cv2.HoughLines(img, 1, np.pi / 180, 150, None, 0, 0)
    #
    list1= []
    list2 = []
    list3 = []
    list4 = []
```

Coding: Seam tracker offline

```
if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a * rho
        y0 = b * rho
        list1.append(x0)
        list2.append(y0)
        list3.append(a)
        list4.append(b)

    list1 = np.array(list1)
    list2 = np.array(list2)
    list3 = np.array(list3)
    list4 = np.array(list4)
    x0 = list1.mean()
    y0 = list2.mean()
    a = list3.mean()
    b = list4.mean()

    pt1 = (int(x0 + 10*(-b)), int(y0 + 10*(a)))
    pt2 = (int(x0 - 10*(-b)), int(y0 - 10*(a)))
    cv2.line(frame, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)

cv2.imshow('video',frame)
cv2.imshow('canny', img)
key = cv2.waitKey(10)
```

```
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()

histg = cv2.calcHist([img2],[0],None,[256],[0,256])
plt.plot(histg)
plt.show()
```

Classification with traditional approach



Convolutions

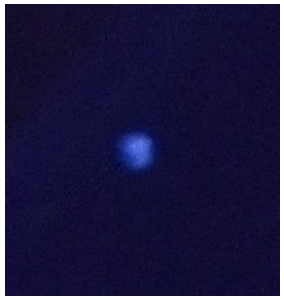
Extract manually features from different filters

- Mean
- Variance
- Number of white pixels
- ...

Rule based classification/ machine learning alg. Like:

- k-NN
- Random forest
- Support Vector Machine
- ...

Classification with traditional approach



Convolutions

Extract manually features from different filters

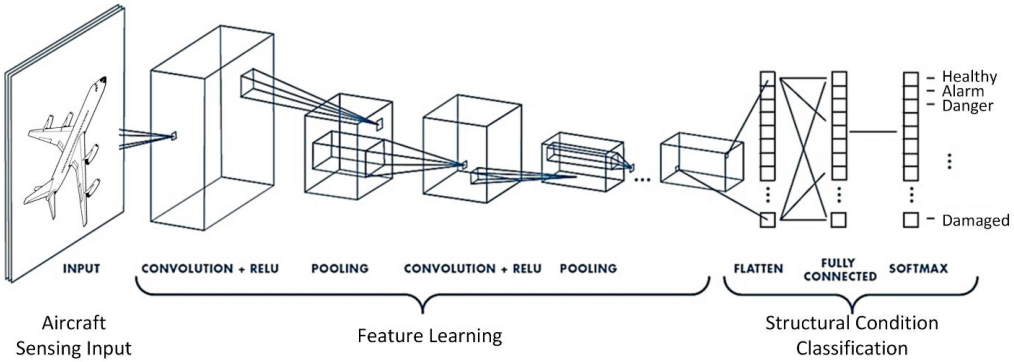
- Mean
- Variance
- Number of white pixels
- ...

Why not apply NN to learn how extract features from images?

Rule based classification/ machine learning alg. Like:

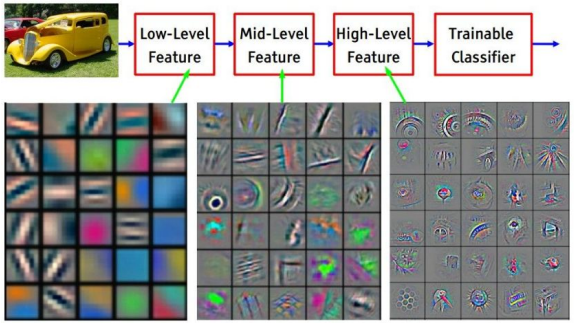
- k-NN
- Random forest
- Support Vector Machine
- ...

Convolutional Neural Network (I)

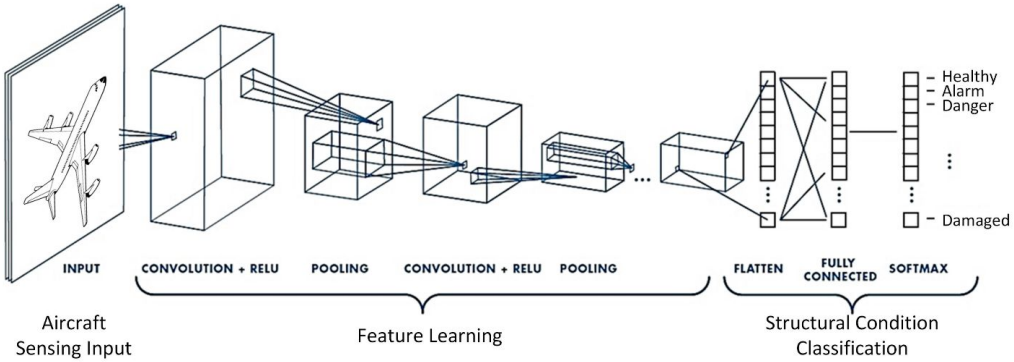


CNN are Networks specialized to learn how extract features from images.

More in deeper you go, more high-level features you extract.

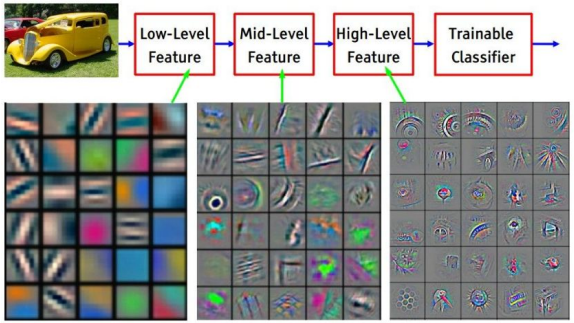


Convolutional Neural Network (I)



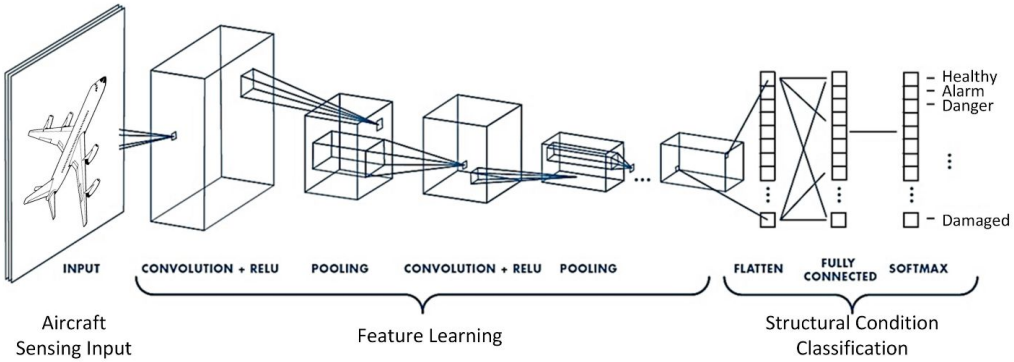
CNN are Networks specialized to learn how extract features from images.

More in deeper you go, more high-level features you extract.



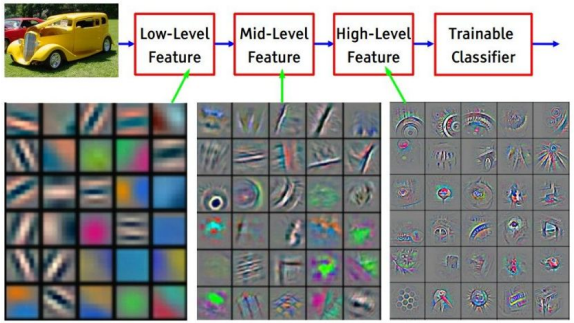
The activation function used in this networks is ReLU

Convolutional Neural Network (I)



CNN are Networks specialized to learn how extract features from images.

More in deeper you go, more high-level features you extract.

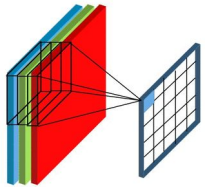


The activation function used in this networks is ReLU

What is pooling and why??

Convolutional Neural Network: brief recap (II)

Convolutional layer



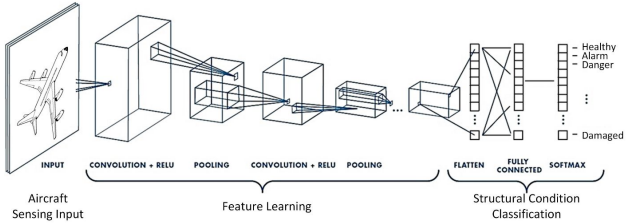
Feature Map

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Max Pooling

Average Pooling

Sum Pooling



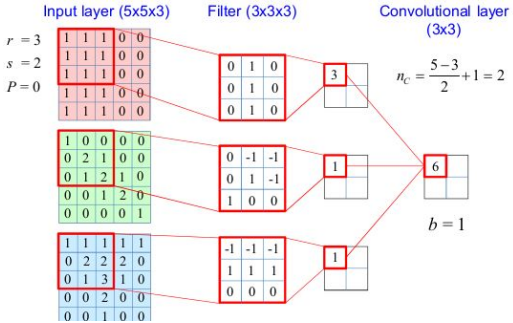
$$d = \frac{n-f+2p}{s} + 1$$

- d output
- n input
- f kernel
- p padding
- s stride

Apply g filters (f,f) from place to g images of size (d,d).

Convolutional Neural Network: how much memory? (III)

In a square filter there are f*f+1 (bias) parameter for a B/W image, and p=3*f*f+1 for colored.



If we use n filters we have for each convolutional layer, considering that pooling has not parameters, p*n parameters. We note that **this number does not depend on the size of the input image.**

- If we work on a b/w image with filter 5x5 and n = 20 we need 520 parameter, that is less then a fully connected layer with 1 hidden layer with 30 neurons with image input of size 28x28 = 23550 (more then 40 times)
- Clearly, full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Network: Pooling benefits? (III)

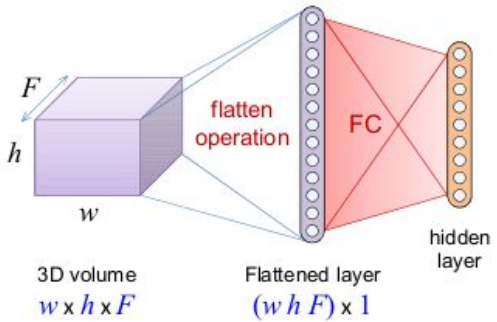
The function of Pooling is to reduce the spatial size of the input.

- It makes the feature map smaller and more manageable.
- It reduces the number of parameters and computations in the network, therefore, controlling overfitting.
- It makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling).
- it allows an almost scale-invariant representation of the image. This is very powerful since we can detect objects in an image no matter where they are located.

Note that we can change the pooling with **larger stride.**

Convolutional Neural Network: Classification (IV)

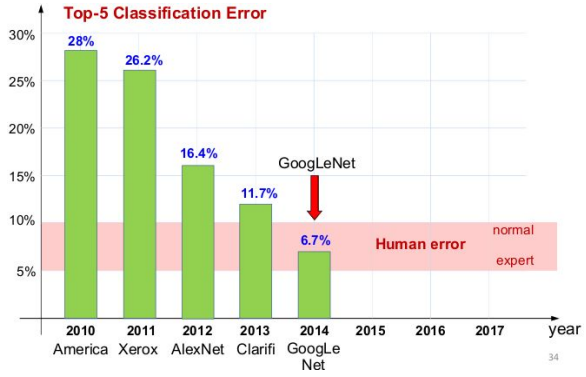
In the last convolutional layer we compress all high-level features in one vector. **Flatten operation**

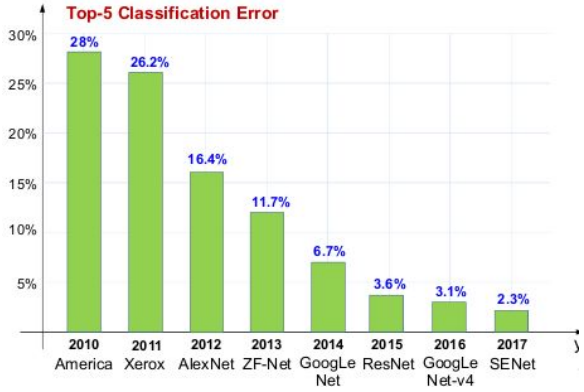


These compressed information are used to train a classifier, that is a feedforward network. If the problem is a multiclass classification the output activation is a **softmax.**

ILSVRC

The ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is a sort of annual Olympics of computer vision, started in 2010, where algorithms compete on complex image recognition problems. ImageNet is a large database containing over 14 million images divided in 20,000 categories annotated to indicate what objects are pictured.

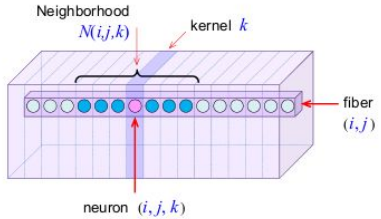




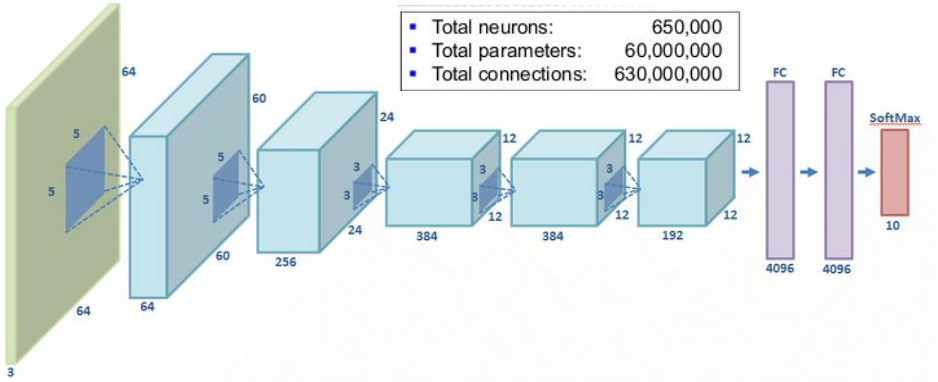
AlexNet (2012)

Multiple convolution layers are stacked before the pooling layer, rather than alternating convolution, activation, and pooling.

- It uses ReLU and a final softmax
- Local Response Normalization (normalization respect neighborhood)



- Dropout during training
- Data augumentation

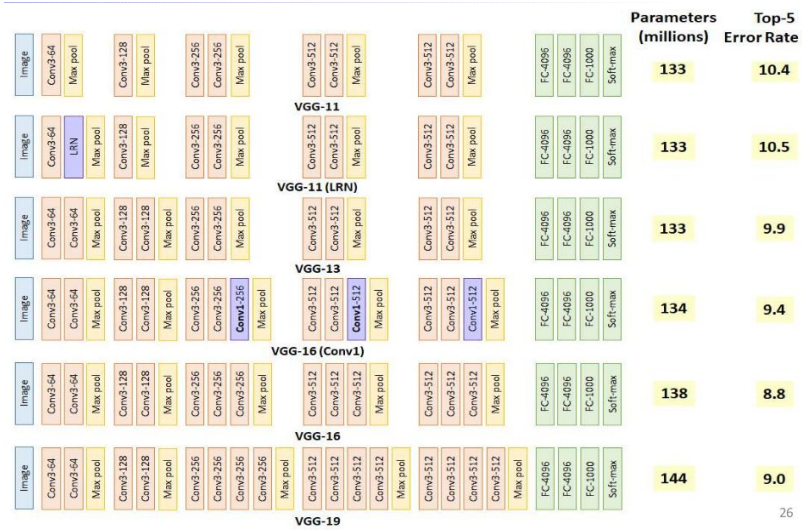


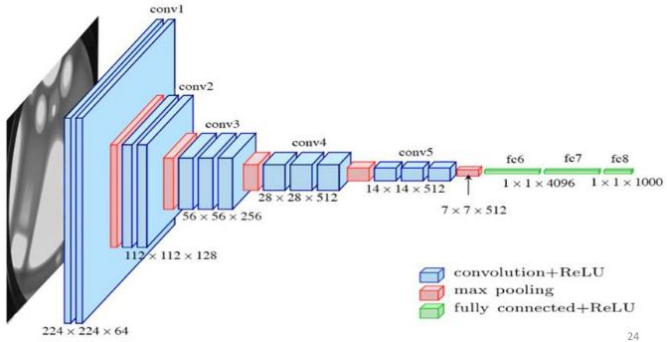
*At that moment, NVIDIA GTX 580 GPU had only 3 GB of Memory, thus the architecture was split into two paths to use two GPUs and was trained for **6 days.***

VGG (I)

Main features:

- Stack multiple conv layers before pooling speed up training
- **deeper**
- ReLU for each conv layer
- Trained on 4 Nvidia Titan Black GPU for 3 weeks



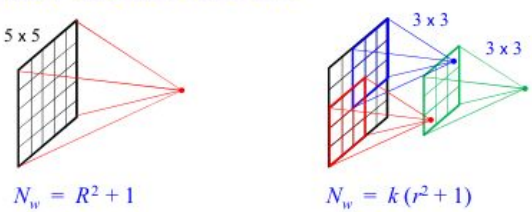


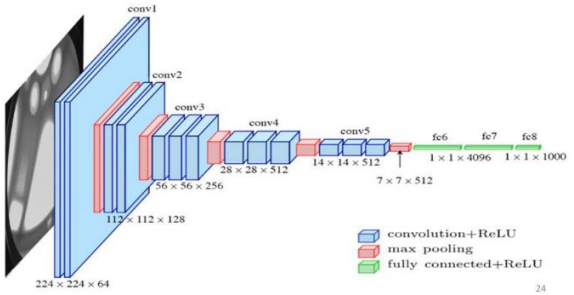
It did not win the 2014 ILSVRC challenge, but it is important for 2 reason:

- Divide by 2 image size and double filters help accuracy
- Stack multiple conv speed up training

VGG (II)

Main idea: Three 3x3 conv layers have a receptive field of size 7x7

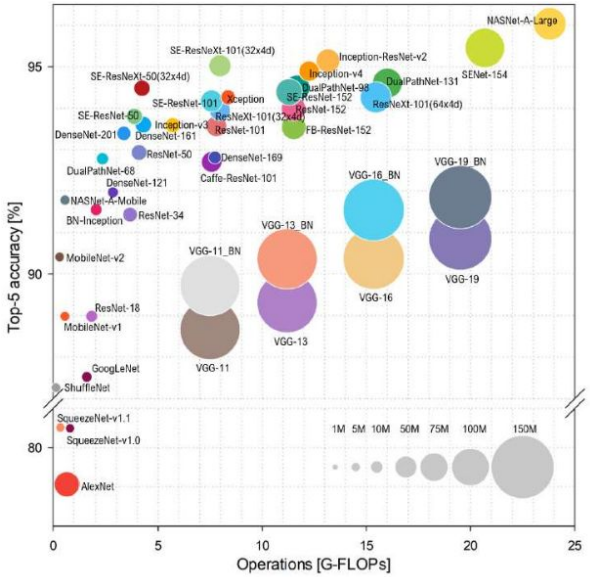




If we want an equivalent R x R filter using an r x r filter, we have to stack **k** layers, where k is given by:

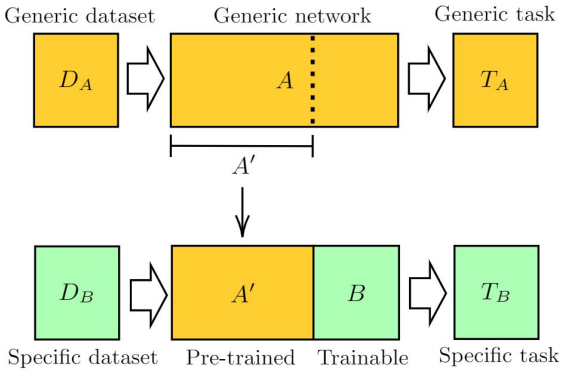
$$R = k(r-1) + 1 \rightarrow k = \frac{R-1}{r-1}$$

k	$R = k\,(r-1)+1$	*weights* R^2+1	*weights* $k\,(r^2+1)$	Diff.	%
2	5	26	20	6	**23 %**
3	7	50	30	20	**40 %**
4	9	82	40	42	**51%**
5	11	122	50	72	**59 %**



Transfer learning

Transfer learning is the process of taking a pre-trained network (using a large dataset) and “fine-tuning” it with your own dataset.

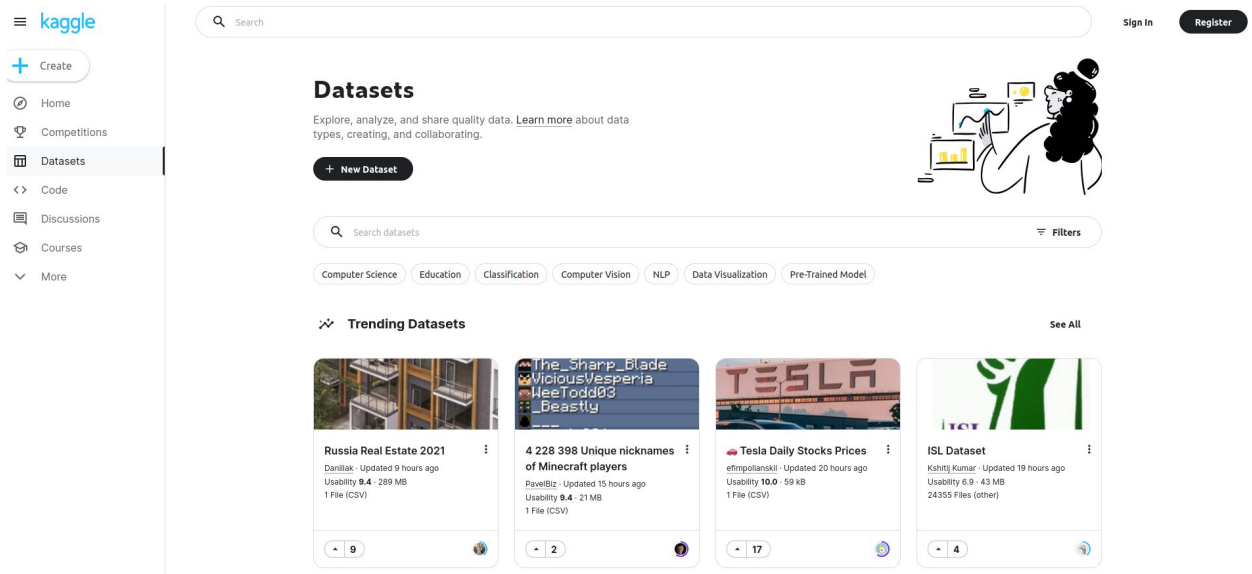


Agenda : Lesson 4

- What’s next? An quick intro to:
  - Recurrent neural networks
  - Reinforcement learning
- Open issue
- Introduction to AI framworks

Kaggle

Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.



https://www.kaggle.com/

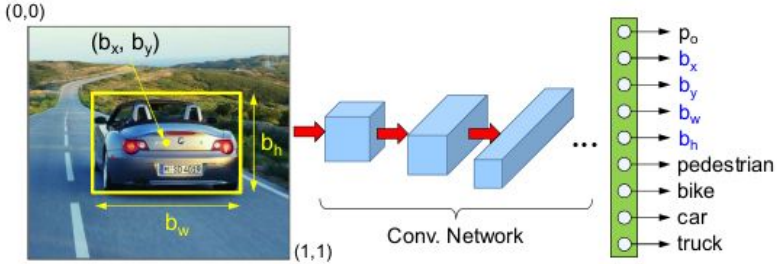
Other applications for CNN

Classification

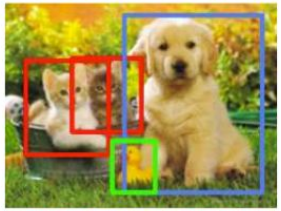
CAT

Classification + Localization

CAT



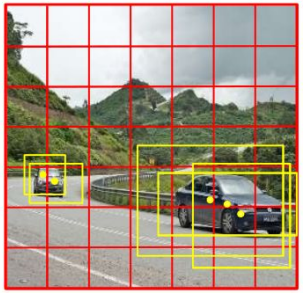
Classification + Detection



CAT, DOG, DUCK

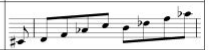
Classification + Segmentation

CAT, DOG, DUCK



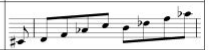
Other applications of deep learning: Recurrent Neural Networks (I)

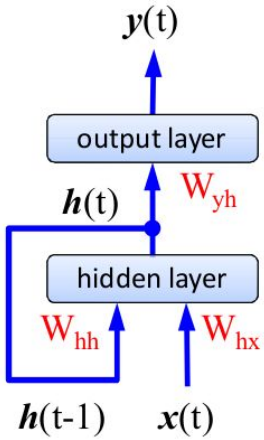
NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

Other applications of deep learning: Recurrent Neural Networks (II)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

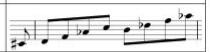
Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

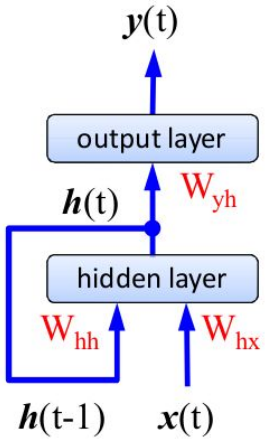


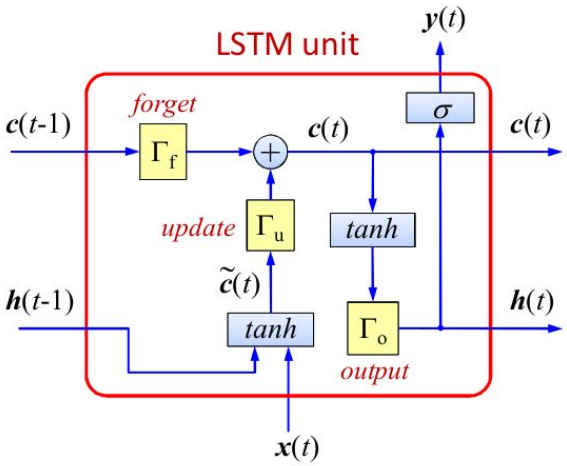
Classical RNN

Other applications of deep learning: Recurrent Neural Networks (III)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AG CCCCTGTGAGGAACTAG 00 1111111111111111 00 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

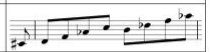


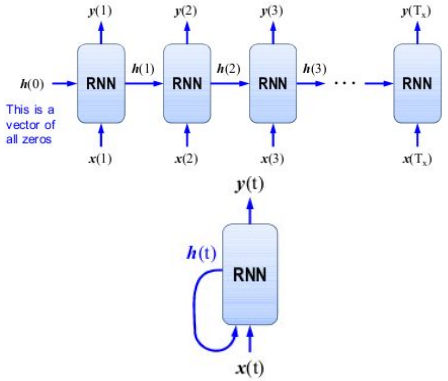


Long-Short Term Memory

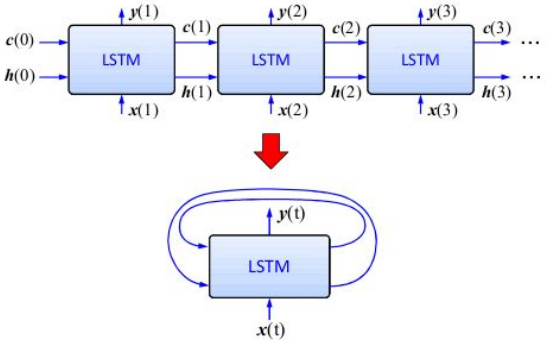
Other applications of deep learning: Recurrent Neural Networks (IV)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AG**CCCCTGTGAGGAACT**AG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron



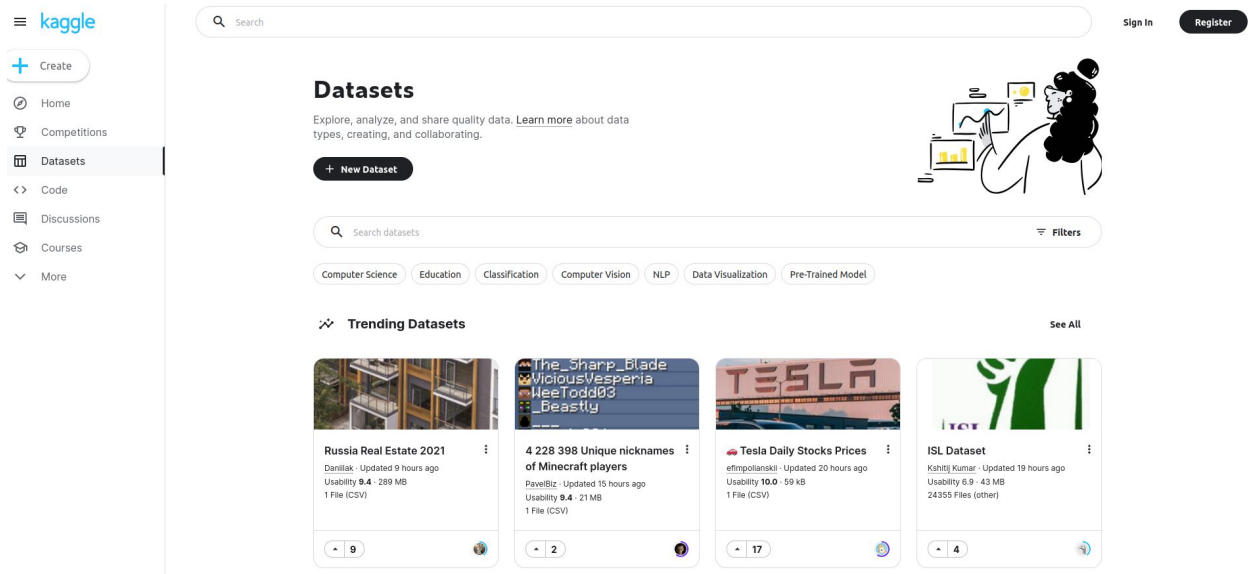
Classical RNN



Long-Short Term Memory

Kaggle

Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.



https://www.kaggle.com/

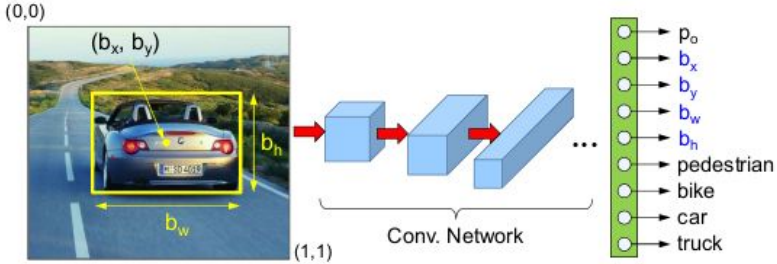
Other applications for CNN

Classification

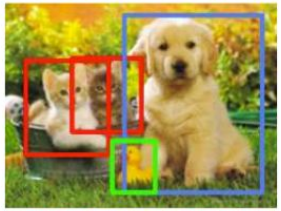
CAT

Classification + Localization

CAT



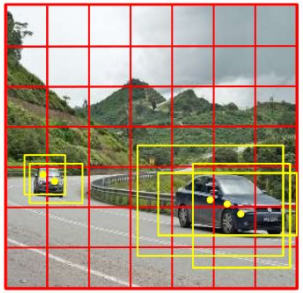
Classification + Detection



CAT, DOG, DUCK

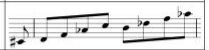
Classification + Segmentation

CAT, DOG, DUCK



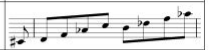
Other applications of deep learning: Recurrent Neural Networks (I)

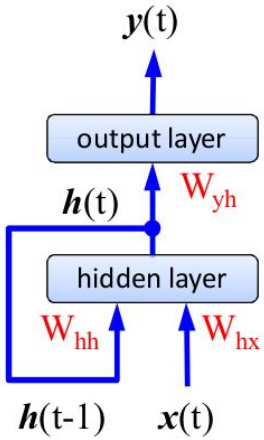
NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

Other applications of deep learning: Recurrent Neural Networks (II)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

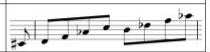
Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

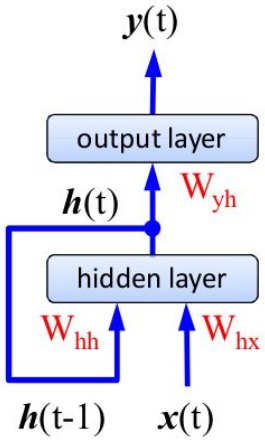


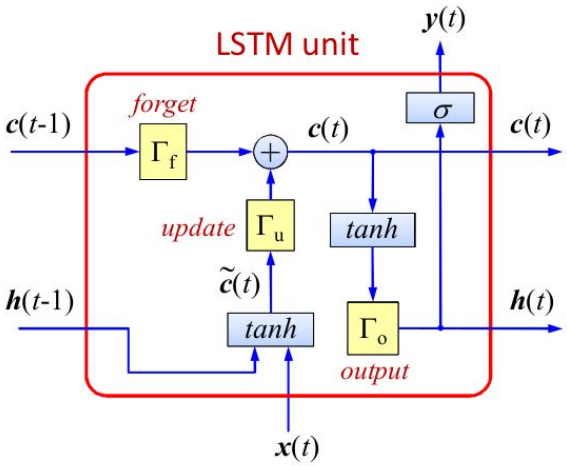
Classical RNN

Other applications of deep learning: Recurrent Neural Networks (III)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AG CCCCTGTGAGGAACTAG 00 1111111111111111 00 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

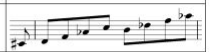


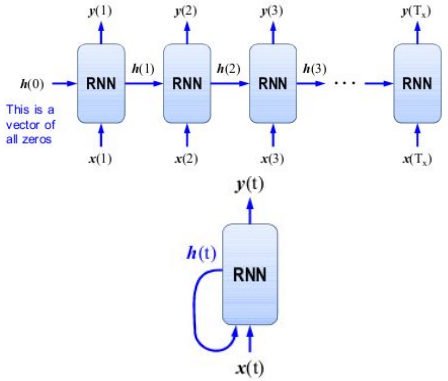


Long-Short Term Memory

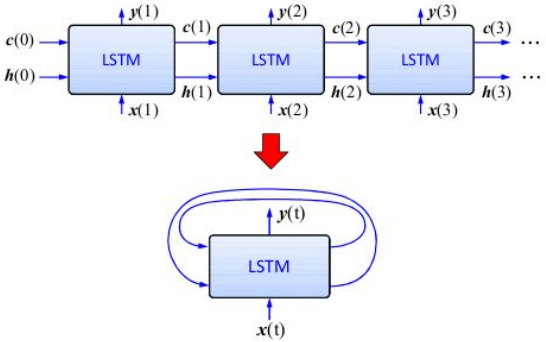
Other applications of deep learning: Recurrent Neural Networks (IV)

NN type	input	output
Classical	x_1	y_1
Classical or Recurrent	x_1, x_2, x_3, x_4	y_1
Recurrent	x_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Recurrent	x_1, x_2, x_3	y_1, y_2, y_3, y_4, y_5

Application	Input	Output
Speech recognition		"It's amazing what neural networks can do."
Music generation	A number	
Sentiment classification	"This movie was annoying and not very interesting."	A number denoting the evaluation rating: ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AG**CCCCTGTGAGGAACT**AG 00111111111111111100 protein
Machine translation	Ti piace questa canzone?	Do you like this song?
Video activity recognition		walking
Name entity recognition	Avatar and Aliens were directed by James Cameron	Avatar and Aliens were directed by James Cameron

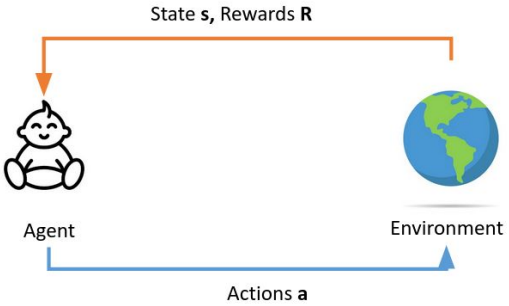


Classical RNN



Long-Short Term Memory

Other applications of deep learning: Reinforcement learning (I)

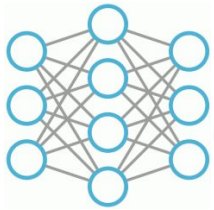


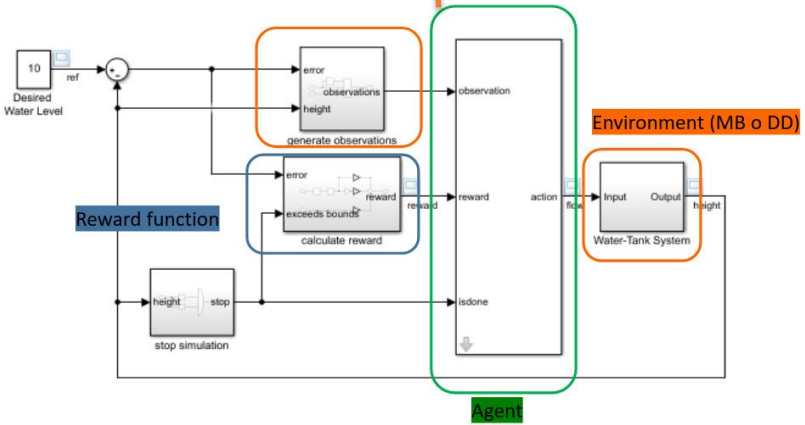
Other applications of deep learning: Reinforcement learning (II)

Other applications of deep learning: Reinforcement learning (III)

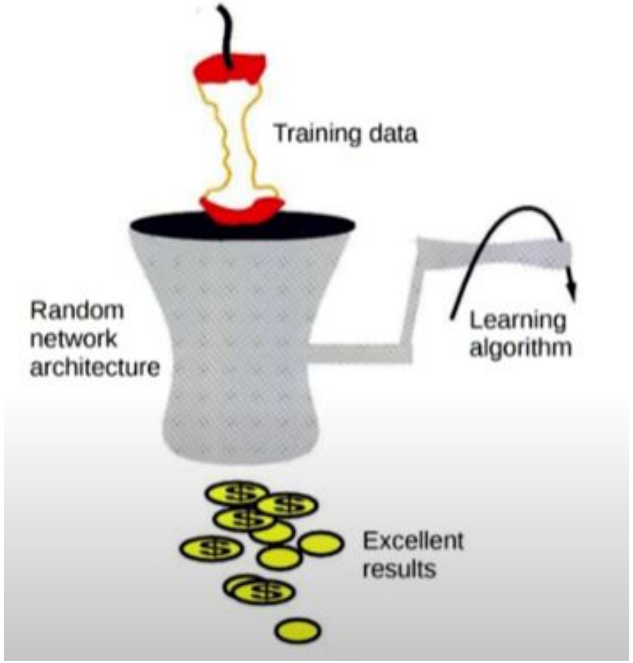

```
Given an env (environment) and an agent:
for episode = 0, ..., MAX_EPISODE do
    state = env.reset()
    agent.reset()
    for t = 0, ..., T do
        action = agent.act(state)
        state, reward = env.step(action)
        agent.update(action, state, reward)
        if env.done() then
            break
        end if
    end for
end for
```

Other applications of deep learning: Reinforcement learning (IV)

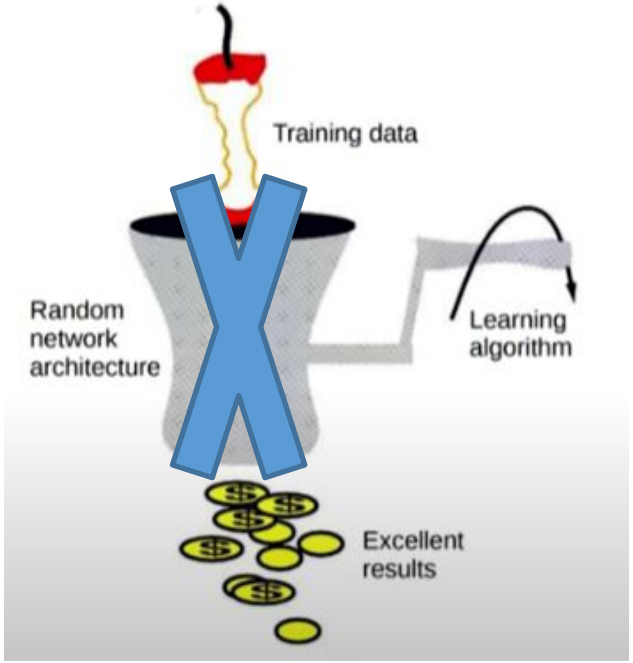




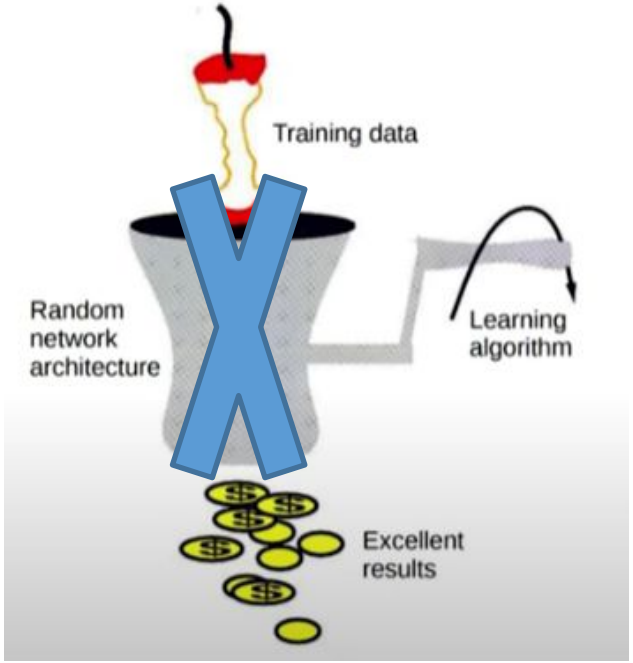
Open iussue (I)

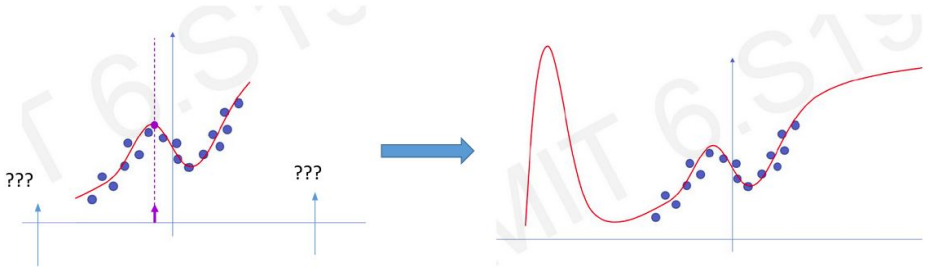


Open issue (I)

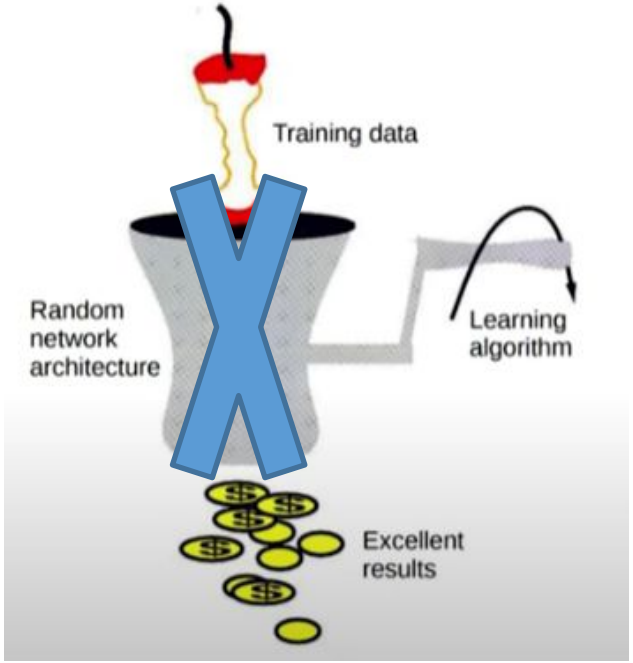

Training data
Random
network
architecture
Learning
algorithm
Excellent
results

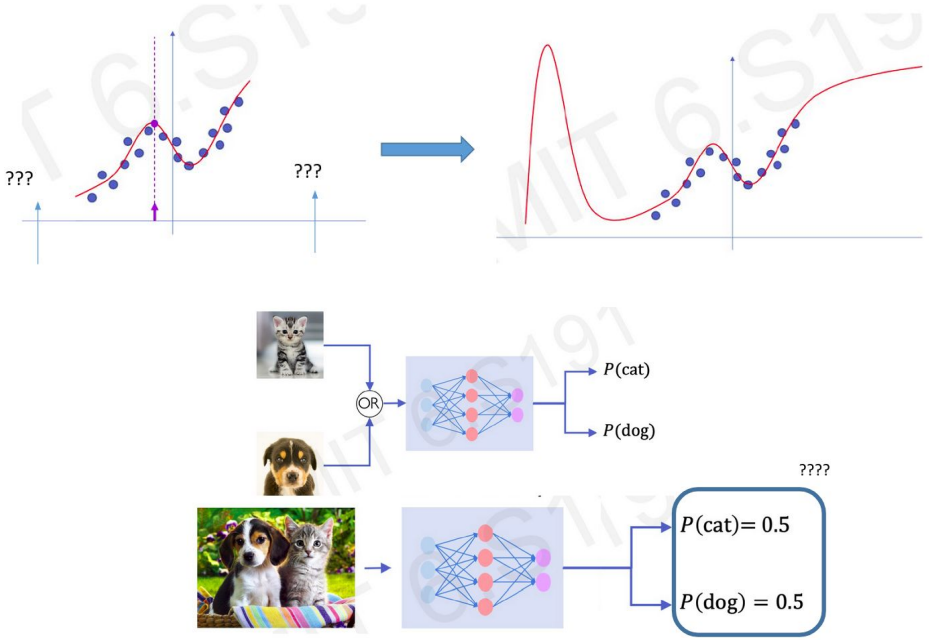
Open issue (II)



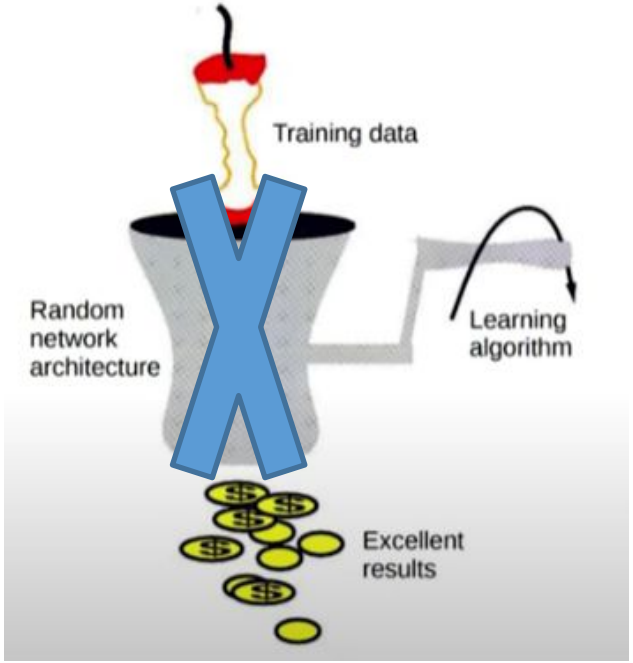


Open issue (III)

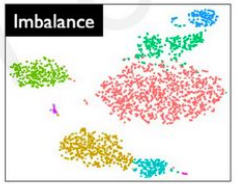




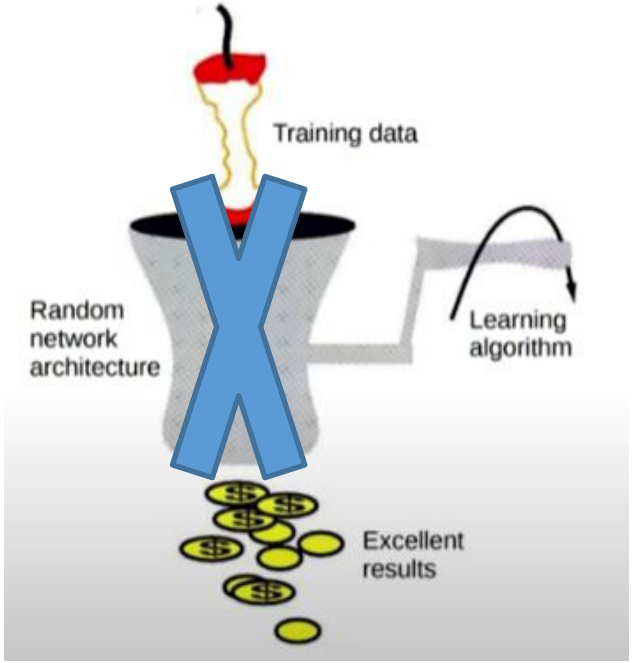
Open issue (IV)



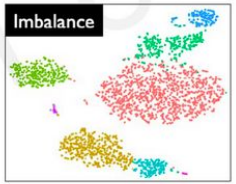
Sparse and/or noisy datasets



Open issue (V)

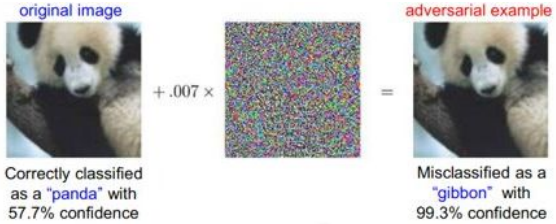


Sparse and/or noisy datasets

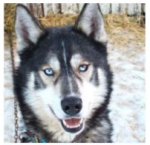


Open issue (VI)

- **Adversarial attack:** DNN are prone to adversarial examples, i.e. specially crafted inputs that look like normal to humans, but cause the AI model to make a wrong prediction.



- **Bias** : a husky was misclassified as a wolf because the network learned to use snow as a feature (all the pictures of wolfs in the training set were taken in the snow



- Similar things can be done with the human vision system, which is particularly specialized to face recognition.

API for DNN modeling (I)

DNNs are typically developed, trained, and inferred by means of specific frameworks (i.e., toolkits, libraries), because:

- They provide a simple high-level interface (mostly in Python) wich simplify the creation of new DNNs using a large set of pre-implemented layers (e.g., convolutions);
- They allow training and inferring DNNs on different devices (e.g., CPUs, GPUs, mobile) by changing a few lines of code;
- The best player in AI spend time and money to develop and update their framework

API for DNN modeling (II)

Caffe (2013) by UC Berkeley

TensorFlow (2015) by Google

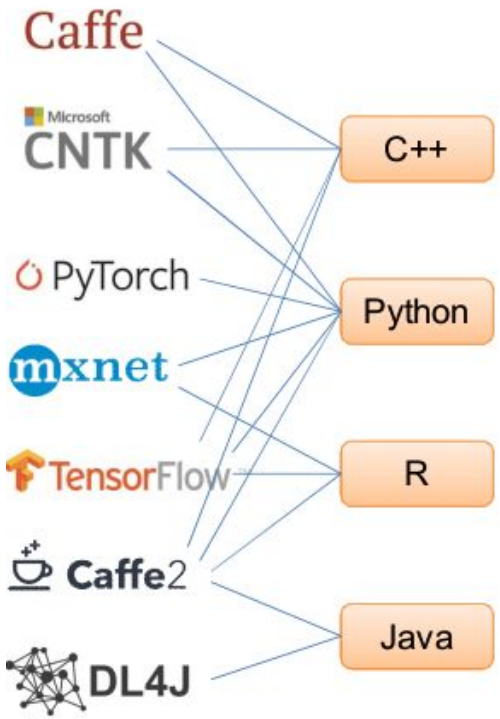
mxnet (2015) by Amazon

Microsoft CNTK (2016) (Computational Network Toolkit) by Microsoft. Now called Microsoft Cognitive Toolkit.

Caffe2 (2017) by Facebook as an evolution of Caffe.

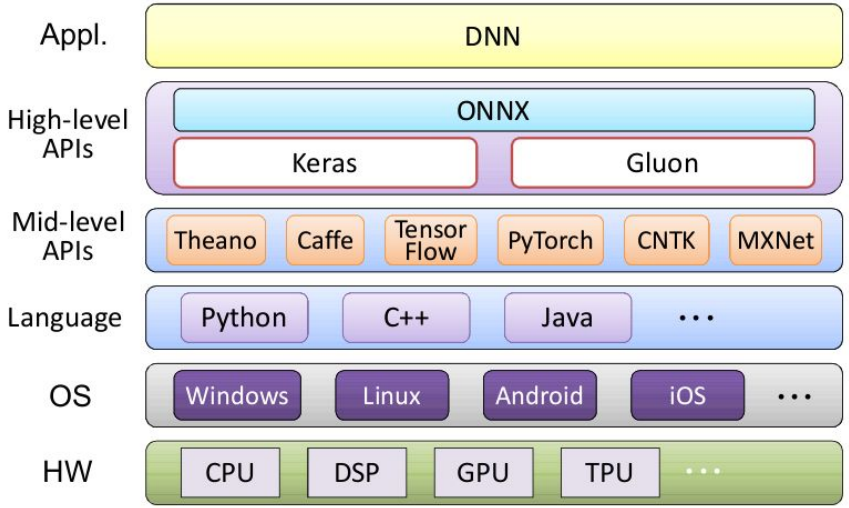
PyTorch (2017) by Facebook. Caffe2 and Pytorch are going to be integrated into a single platform.

DL4J (2017) (Deep Learning for Java) by Skymind as a deep learning library for Java and Scala.



API for DNN modeling (III)

- **Keras** is an open-source software library that provides a Python interface for artificial neural networks
- Released in 2016 by Google
- It allows creating a DNN by stacking layers without specifying math operations, but only layer types.
- It's on top of **Tensorflow, THE MOST USED FRAMEWORK**

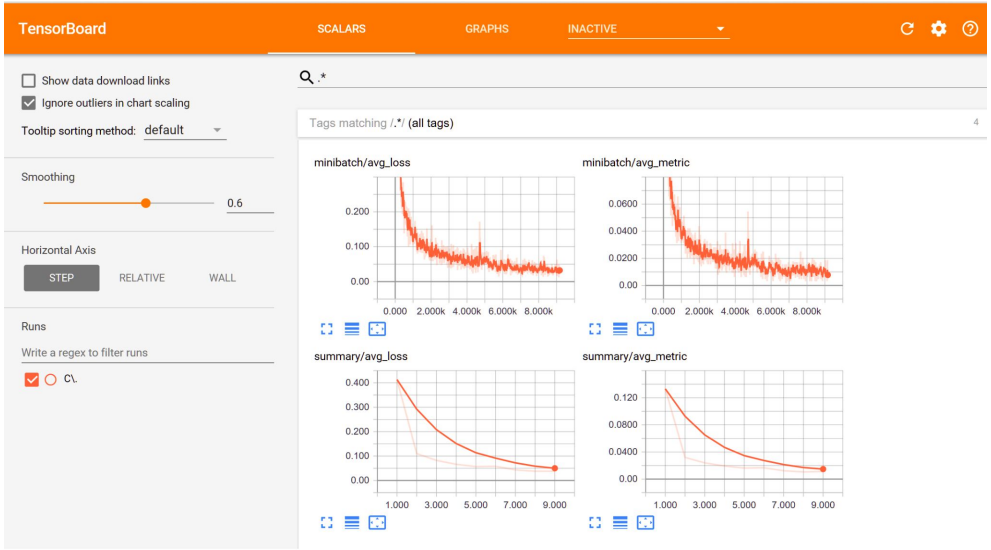


API for DNN modeling (IV)

TensorFlow

	Languages	Tutorial Material	CNN modeling	RNN modeling	Easy-to-use API	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	NO	YES
Tensor-Flow	Python, C++	+++	+++	++	+++	++	YES	YES
Pytorch	Python, C++	+	+++	++	++	+++	YES	NO
Caffe	Python, C++	+	+++	NO	+	+	YES	NO
MXNet	Python, R, Julia, Scala	++	++	+	++	++	YES	YES
CNTK	Python, C++	+	+	+++	+	++	YES	YES
DL4J	Java, Scala	+++	+++	+++	++	++	YES	YES

- Since 2017, the Keras API is integrated in TensorFlow and can be used as tf.keras (in a Python environment)
- It has a lot of useful tools, like **tensorboard**



https://www.tensorflow.org/ https://www.tensorflow.org/resources/tools

Agenda : Lesson 5

- DNN Modeling with keras
- Demos
  - ShallowNet with TF
  - Welding defect detection with TF
  - Transfer learning for object classification

API for DNN modeling (I)

DNNs are typically developed, trained, and inferred by means of specific frameworks (i.e., toolkits, libraries), because:

- They provide a simple high-level interface (mostly in Python) wich simplify the creation of new DNNs using a large set of pre-implemented layers (e.g., convolutions);
- They allow training and inferring DNNs on different devices (e.g., CPUs, GPUs, mobile) by changing a few lines of code;
- The best player in AI spend time and money to develop and update their framework

API for DNN modeling (II)

Caffe (2013) by UC Berkeley

TensorFlow (2015) by Google

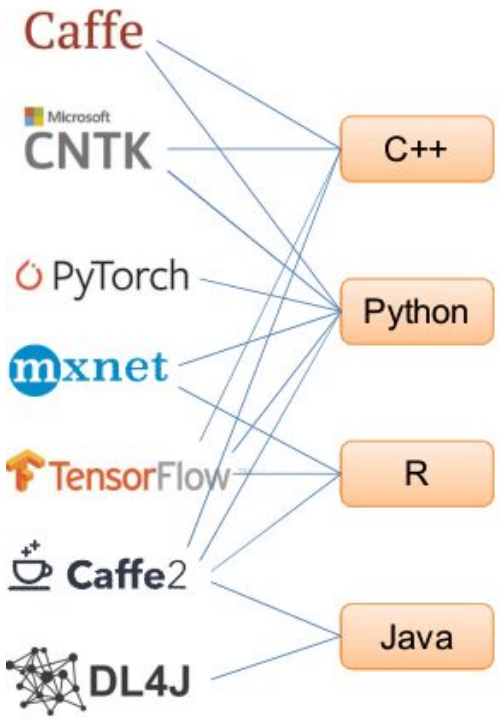
mxnet (2015) by Amazon

Microsoft CNTK (2016) (Computational Network Toolkit) by Microsoft. Now called Microsoft Cognitive Toolkit.

Caffe2 (2017) by Facebook as an evolution of Caffe.

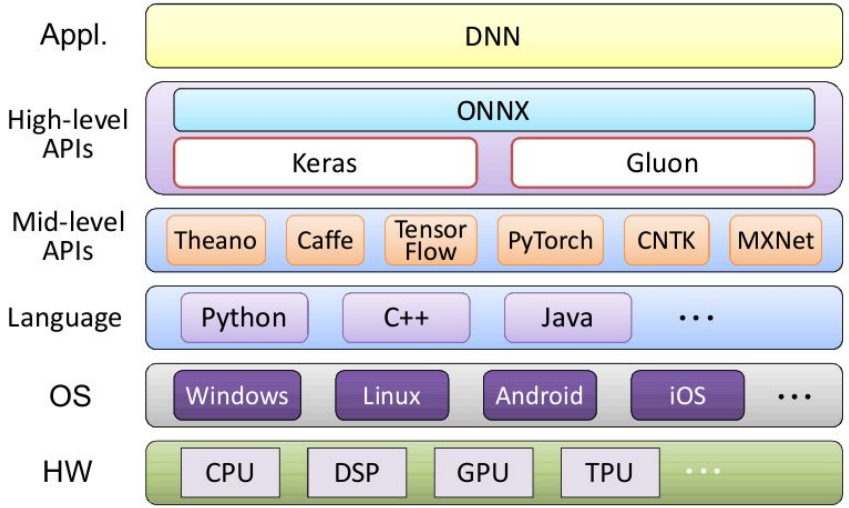
PyTorch (2017) by Facebook. Caffe2 and Pytorch are going to be integrated into a single platform.

DL4J (2017) (Deep Learning for Java) by Skymind as a deep learning library for Java and Scala.



API for DNN modeling (III)

- **Keras** is an open-source software library that provides a Python interface for artificial neural networks
- Released in 2016 by Google
- It allows creating a DNN by stacking layers without specifying math operations, but only layer types.
- It's on top of **Tensorflow, THE MOST USED FRAMEWORK**

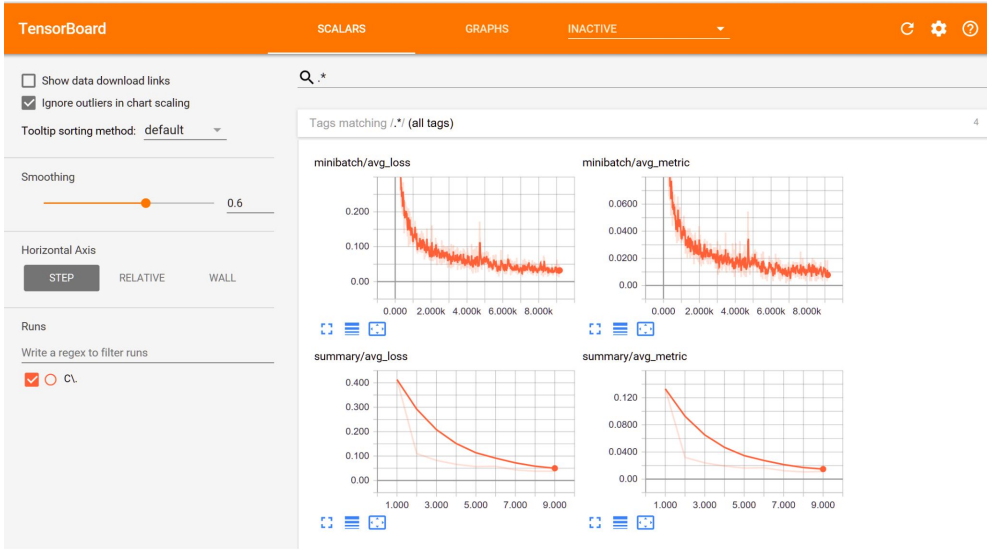


API for DNN modeling (IV)

TensorFlow

	Languages	Tutorial Material	CNN modeling	RNN modeling	Easy-to-use API	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	NO	YES
Tensor-Flow	Python, C++	+++	+++	++	+++	++	YES	YES
Pytorch	Python, C++	+	+++	++	++	+++	YES	NO
Caffe	Python, C++	+	+++	NO	+	+	YES	NO
MXNet	Python, R, Julia, Scala	++	++	+	++	++	YES	YES
CNTK	Python, C++	+	+	+++	+	++	YES	YES
DL4J	Java, Scala	+++	+++	+++	++	++	YES	YES

- Since 2017, the Keras API is integrated in TensorFlow and can be used as tf.keras (in a Python environment)
- It has a lot of useful tools, like **tensorboard**



https://www.tensorflow.org/ https://www.tensorflow.org/resources/tools

API for DNN modeling (V)

https://www.tensorflow.org/api_docs/python/tf

- tf.keras
  - Overview
  - Input
  - Model
  - Sequential
  - activations
  - applications
  - backend
  - callbacks
  - constraints
  - datasets
  - dtensor
  - estimator
  - experimental
  - initializers
  - layers
  - losses
  - metrics
  - mixed_precision
  - models
  - optimizers
  - preprocessing
  - regularizers
  - utils
  - wrappers

API for DNN modeling (V)

https://www.tensorflow.org/api_docs/python/tf

- tf.keras
  - Overview
  - Input
  - Model
  - Sequential
- activations
- applications
- backend
- callbacks
- constraints
- datasets
- dtensor
- estimator
- experimental
- initializers
- layers
- losses
- metrics
- mixed_precision
- models
- optimizers
- preprocessing
- regularizers
- utils
- wrappers

Classes

`class Model`: `Model` groups layers into an object with training and inf

`class Sequential`: `Sequential` groups a linear stack of layers into a

API for DNN modeling (V)

https://www.tensorflow.org/api_docs/python/tf

Classes

`class Model`: `Model` groups layers into an object with training and inf

`class Sequential`: `Sequential` groups a linear stack of layers into a

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
# Afterwards, we do automatic shape inference:
model.add(tf.keras.layers.Dense(4))

# This is identical to the following:
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(16,)))
model.add(tf.keras.layers.Dense(8))
```

Sequential model:

- Add layers in sequential way
- Good to start with TF
- For simple architectures

API for DNN modeling (V)

https://www.tensorflow.org/api_docs/python/tf

Classes

`class Model`: `Model` groups layers into an object with training and inf

`class Sequential`: `Sequential` groups a linear stack of layers into a

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
# Afterwards, we do automatic shape inference:
model.add(tf.keras.layers.Dense(4))

# This is identical to the following:
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(16,)))
model.add(tf.keras.layers.Dense(8))
```

Sequential model:

- Add layers in sequential way
- Good to start with TF
- For simple architectures

API for DNN modeling (V)

https://www.tensorflow.org/api_docs/python/tf

Classes

`class Model`: `Model` groups layers into an object with training and inf

`class Sequential`: `Sequential` groups a linear stack of layers into a

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
# Afterwards, we do automatic shape inference:
model.add(tf.keras.layers.Dense(4))

# This is identical to the following:
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(16,)))
model.add(tf.keras.layers.Dense(8))
```

Sequential model:

- Add layers in sequential way
- Good to start with TF
- For simple architectures

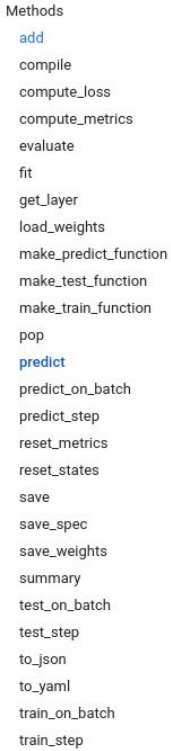
```
mport tensorflow as tf

nputs = tf.keras.Input(shape=(3,))
 = tf.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
utputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
odel = tf.keras.Model(inputs=inputs, outputs=outputs)
```

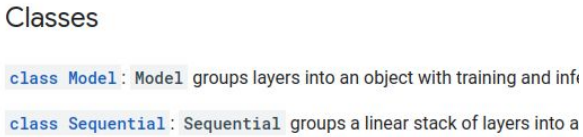
I/O model:

- Define Input layer
- “convolute” a layer “x” with “input” with ()
- define the “output”
- For more complex architectures

API for DNN modeling (VI)

https://www.tensorflow.org/api_docs/python/tf



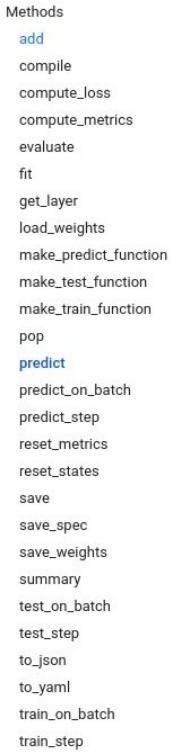


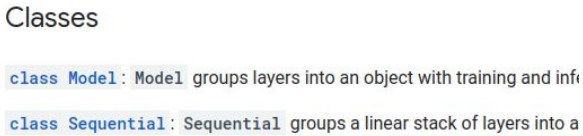
- Model is a class. There are a lot of **methods.**
  - **compile** to associate to prepare model for training

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(),
                       tf.keras.metrics.FalseNegatives()])
```

API for DNN modeling (VII)

https://www.tensorflow.org/api_docs/python/tf



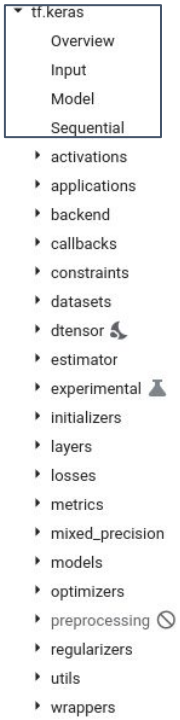


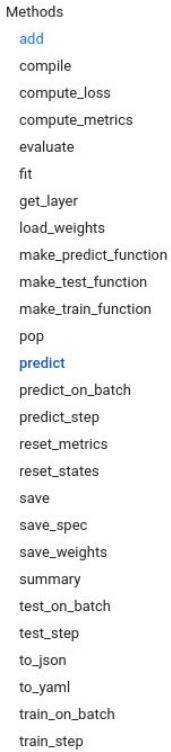
- Model is a class. There are a lot of **methods.**
  - compile to associate to prepare model for training
  - **fit** to start training

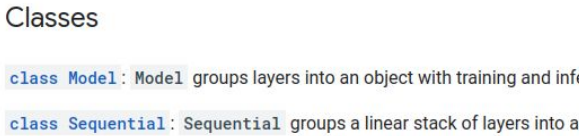
```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

API for DNN modeling (VIII)

https://www.tensorflow.org/api_docs/python/tf







- Model is a class. There are a lot of **methods.**
  - compile to associate to prepare model for training
  - fit to start training
  - **summary** prints a string summary of the network.
  - **save_weights**
  - **predict** for inference

```
predict(
    x,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

```
save_weights(
    filepath, overwrite=True, save_format=None, options=None
)
```

API for DNN modeling (IX)

https://www.tensorflow.org/api_docs/python/tf/keras/layers

Module: tf.keras.optimizers

Classes

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm.

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

`class Nadam` : Optimizer that implements the NAdam algorithm.

`class Optimizer` : Base class for Keras optimizers.

`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Gradient descent (with momentum) optimizer.

Module: tf.keras.regularizers

`class L1` : A regularizer that applies a L1 regularization penalty.

`class L1L2` : A regularizer that applies both L1 and L2 regularization penalties.

`class L2` : A regularizer that applies a L2 regularization penalty.

`class Regularizer` : Regularizer base class.

`class l1` : A regularizer that applies a L1 regularization penalty.

`class l2` : A regularizer that applies a L2 regularization penalty.

API for DNN modeling (X)

https://www.tensorflow.org/api_docs/python/tf/keras/layers

Module: tf.keras.initializers

`class Constant`: Initializer that generates tensors with constant values.

`class GlorotNormal`: The Glorot normal initializer, also called Xavier normal initializer.

`class GlorotUniform`: The Glorot uniform initializer, also called Xavier uniform initializer.

`class HeNormal`: He normal initializer.

`class HeUniform`: He uniform variance scaling initializer.

`class Identity`: Initializer that generates the identity matrix.

`class Initializer`: Initializer base class: all Keras initializers inherit from this class.

`class LecunNormal`: Lecun normal initializer.

`class LecunUniform`: Lecun uniform initializer.

`class Ones`: Initializer that generates tensors initialized to 1.

Module: tf.keras.losses

`class BinaryCrossentropy`: Computes the cross-entropy loss between true labels and predicted labels.

`class BinaryFocalCrossentropy`: Computes the focal cross-entropy loss between true labels and predictions.

`class CategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between labels and predictions.

`class Hinge`: Computes the hinge loss between `y_true` and `y_pred`.

`class Huber`: Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss`: Loss base class.

`class MeanAbsoluteError`: Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError`: Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError`: Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError`: Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson`: Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction`: Types of loss reduction.

`class SparseCategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge`: Computes the squared hinge loss between `y_true` and `y_pred`.

API for DNN modeling (X)

https://www.tensorflow.org/api_docs/python/tf/keras/layers

Module: tf.keras.initializers

`class Constant` : Initializer that generates tensors with constant values.

`class GlorotNormal` : The Glorot normal initializer, also called Xavier normal initializer.

`class GlorotUniform` : The Glorot uniform initializer, also called Xavier uniform initializer.

`class HeNormal` : He normal initializer.

`class HeUniform` : He uniform variance scaling initializer.

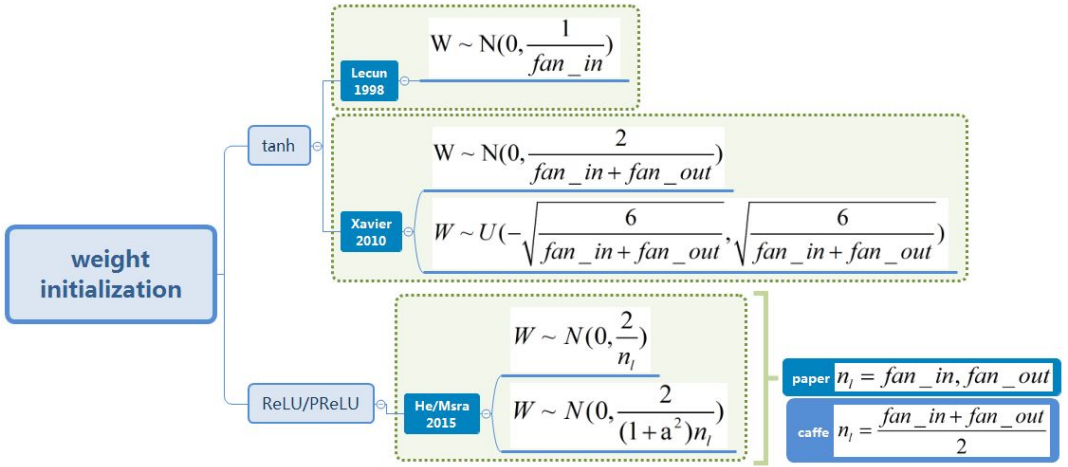
`class Identity` : Initializer that generates the identity matrix.

`class Initializer` : Initializer base class: all Keras initializers inherit from this class.

`class LecunNormal` : Lecun normal initializer.

`class LecunUniform` : Lecun uniform initializer.

`class Ones` : Initializer that generates tensors initialized to 1.



API for DNN modeling (XI)

https://www.tensorflow.org/api_docs/python/tf

- Choose architecture
- Select optimizer
- create model
- compile model
- train it with training and validation datasets
- test DNN with predict function

```
initializer = tf.keras.initializers.GlorotNormal()

# State as input
state_input = tf.keras.layers.Input(shape=(self.num_states) )
state_out_critic = tf.keras.layers.Dense(128,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),
kernel_initializer= initializer)(state_input)

# Action as input
action_input = tf.keras.layers.Input(shape=(self.num_actions))

action_out_critic = tf.keras.layers.Dense(128,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),
kernel_initializer= initializer)(action_input)

# Concatening 2 networks
concat = tf.keras.layers.Concatenate()(
    [state_out_critic, action_out_critic]
    )

concat = tf.keras.layers.Dense(64,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),
kernel_initializer= initializer)(concat)

out = tf.keras.layers.Dense(32,
    activation=tf.keras.layers.LeakyReLU(alpha=0.1),
    kernel_initializer= initializer,
    kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1e-5, l2=1e-4))(concat)

# Predicted Q(s,a)
outputs = tf.keras.layers.Dense(1)(out)
```

```
critic_optimizer = tf.keras.optimizers.Adam(critic_lr)
```

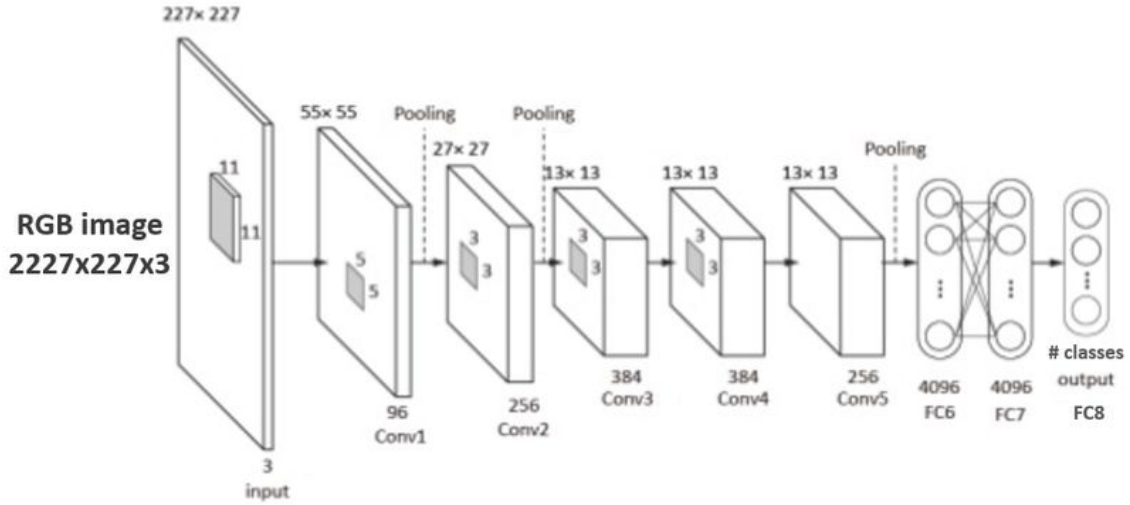
```
tf.keras.Model([state_input, action_input], outputs)
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(),
                       tf.keras.metrics.FalseNegatives()])
```

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

API for DNN modeling (XII)

```
class AlexNet(Sequential):
    def __init__(self, input_shape, num_classes):
        super().__init__()

        self.add(Conv2D(96, kernel_size=(11,11), strides= 4,
                        padding= 'valid', activation= 'relu',
                        input_shape= input_shape,
                        kernel_initializer= 'he_normal'))
        self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                              padding= 'valid', data_format= None))

        self.add(Conv2D(256, kernel_size=(5,5), strides= 1,
                        padding= 'same', activation= 'relu',
                        kernel_initializer= 'he_normal'))
        self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                              padding= 'valid', data_format= None))

        self.add(Conv2D(384, kernel_size=(3,3), strides= 1,
                        padding= 'same', activation= 'relu',
                        kernel_initializer= 'he_normal'))

        self.add(Conv2D(384, kernel_size=(3,3), strides= 1,
                        padding= 'same', activation= 'relu',
                        kernel_initializer= 'he_normal'))

        self.add(Conv2D(256, kernel_size=(3,3), strides= 1,
                        padding= 'same', activation= 'relu',
                        kernel_initializer= 'he_normal'))

        self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                              padding= 'valid', data_format= None))

        self.add(Flatten())
        self.add(Dense(4096, activation= 'relu'))
        self.add(Dense(4096, activation= 'relu'))
        self.add(Dense(1000, activation= 'relu'))
        self.add(Dense(num_classes, activation= 'softmax'))

        self.compile(optimizer= tf.keras.optimizers.Adam(0.001),
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])

model = AlexNet((227, 227, 3), num_classes)
```



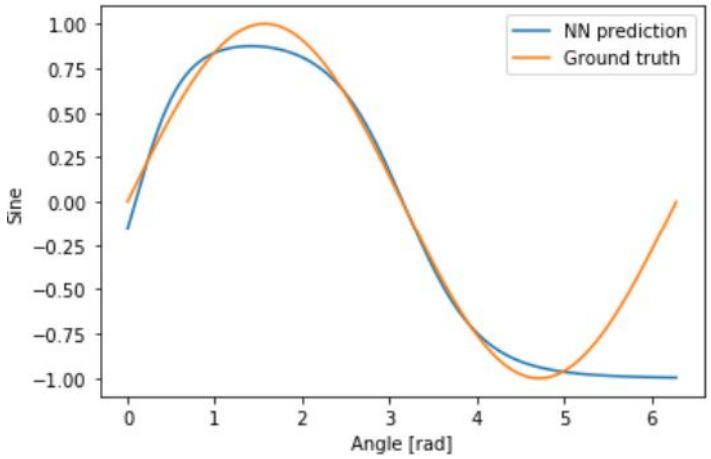
Demo : Generalization of DNN with TF

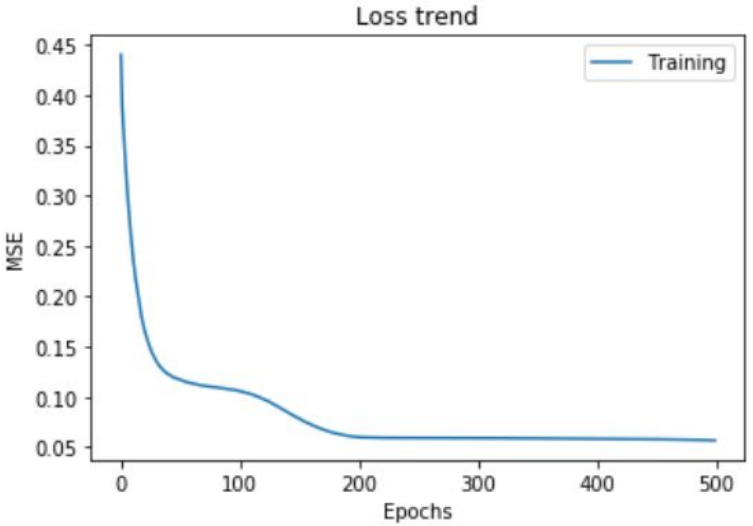
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

#Prepare data
x = np.linspace(0, 360/57.3, 1000)
y = np.sin(x)

#Create network architecture

inp = tf.keras.layers.Input((1,1))
h = tf.keras.layers.Dense(100, activation = tf.nn.tanh)(inp)
out = tf.keras.layers.Dense(1, activation = tf.nn.tanh)(h)

model = tf.keras.Model(inp, out)

#Compile model with loss function and optimizer

model.compile(optimizer = tf.keras.optimizers.RMSprop(1e-4),
              loss = "mse",
              metrics = "accuracy")

#Train
history = model.fit(x, y, epochs = 500)
```

```python
#Plot results
print("[INFO] plotting resume of training...")

yn = model.predict(x)
yn = yn.reshape((yn.shape[0], 1))

plt.plot(x, yn, label = "NN prediction")
plt.plot(x,y, label = "Ground truth")
plt.xlabel("Angle [rad]")
plt.ylabel("Sine")
plt.legend()
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.title('Loss trend')
plt.ylabel('MSE')
plt.xlabel('Epochs')
plt.legend(['Training'], loc='upper right')
plt.show()
```

Demo : Generalization of DNN with TF

alpha = 1e-4, 1 hidden layer with 100 neurons

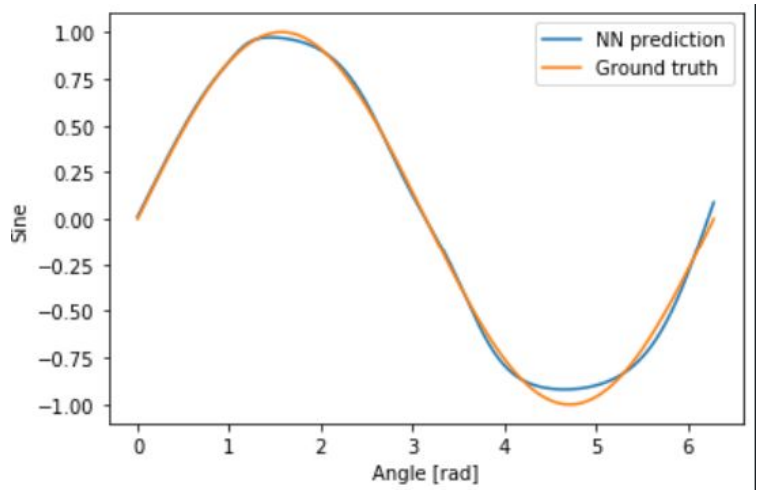


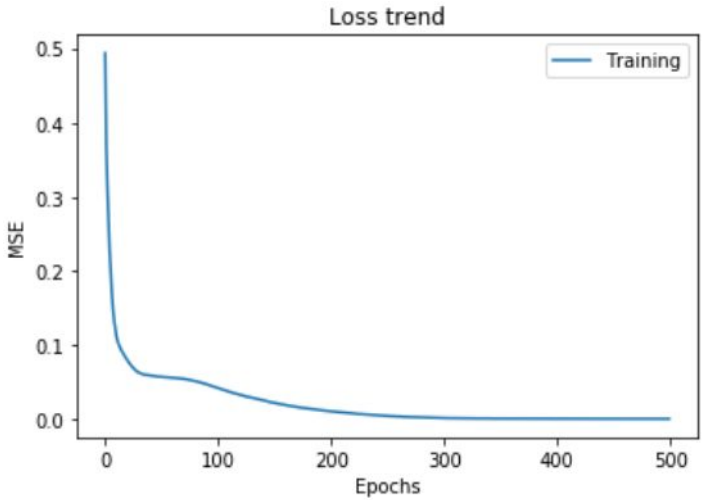


Demo : Generalization of DNN with TF

alpha = 1e-4, 2 hidden layer with 100 neurons

```
inp = tf.keras.layers.Input((1,1))
h = tf.keras.layers.Dense(100, activation = tf.nn.tanh)(inp)
h = tf.keras.layers.Dense(100, activation = tf.nn.tanh)(h)
out = tf.keras.layers.Dense(1, activation = tf.nn.tanh)(h)
```

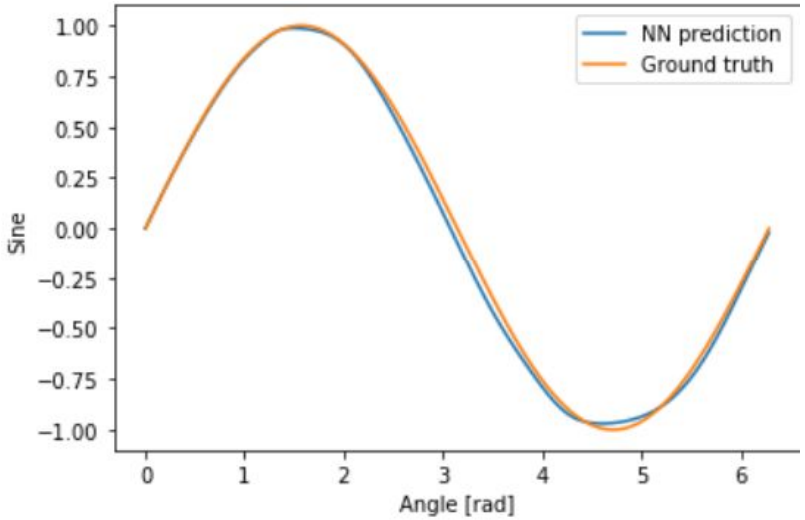


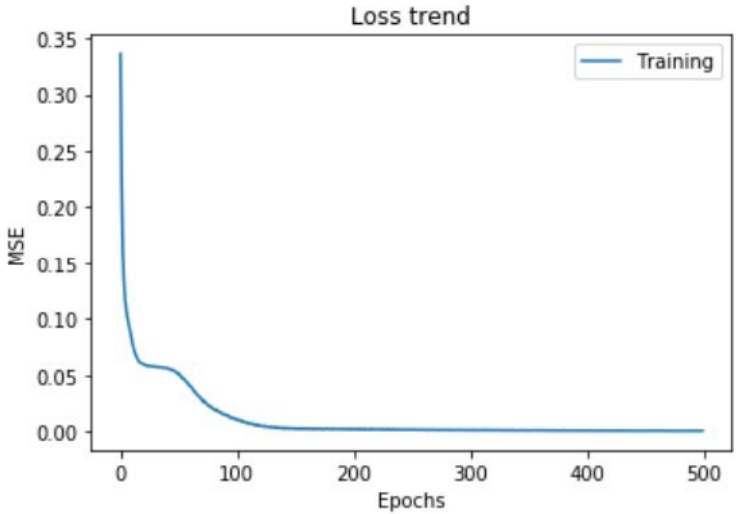


Demo : Generalization of DNN with TF

alpha = 3e-4, 2 hidden layer with 100 neurons

```
inp = tf.keras.layers.Input((1,1))
h = tf.keras.layers.Dense(100, activation = tf.nn.tanh)(inp)
h = tf.keras.layers.Dense(100, activation = tf.nn.tanh)(h)
out = tf.keras.layers.Dense(1, activation = tf.nn.tanh)(h)
```





Demo : Defect detection in welding using DNN

Exercise for home (1 for person)

https://www.kaggle.com/datasets/sudalairajkumar/novel-corona-virus-2019-dataset

https://www.kaggle.com/datasets/prathamtripathi/drug-classification

https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

https://www.kaggle.com/datasets/zalando-research/fashionmnist

https://www.kaggle.com/datasets/gpreda/chinese-mnist

https://www.kaggle.com/code/mohamedzayton/drug-classification-rf-nn#Building-Model-1--%3E-RF

Demo : Transfer learning for flower detection