Tests & Features

IML

Cédric Buche

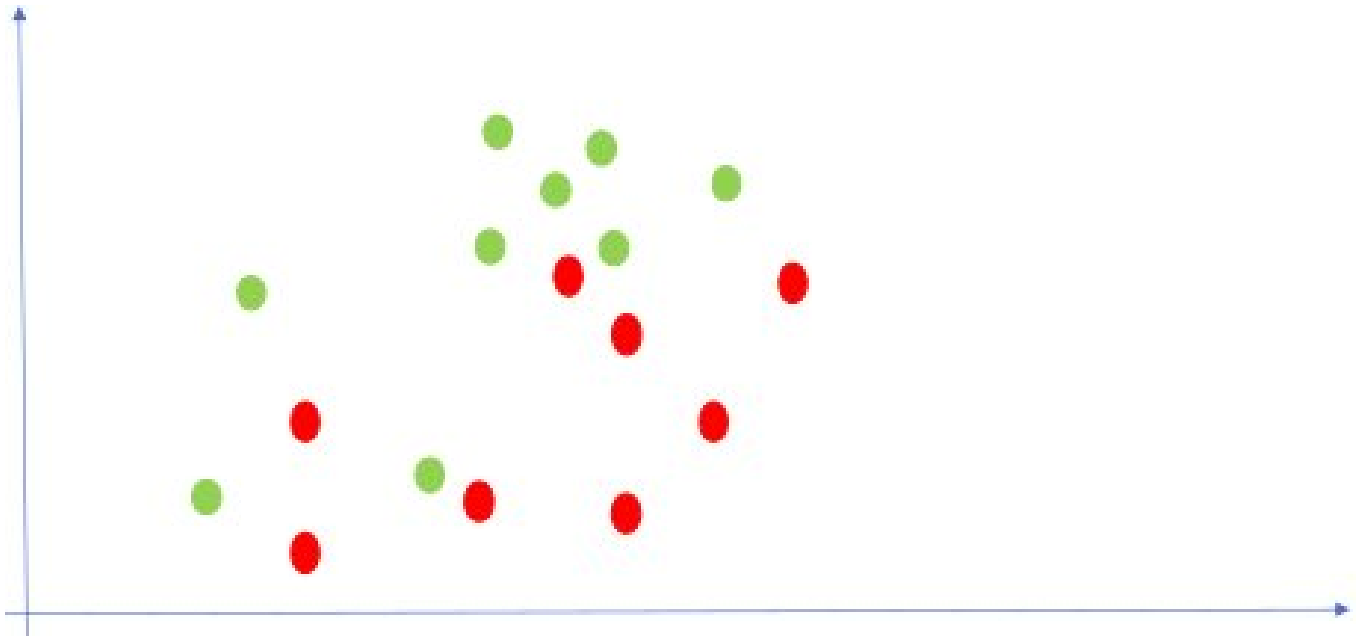
ENIB

29 août 2019

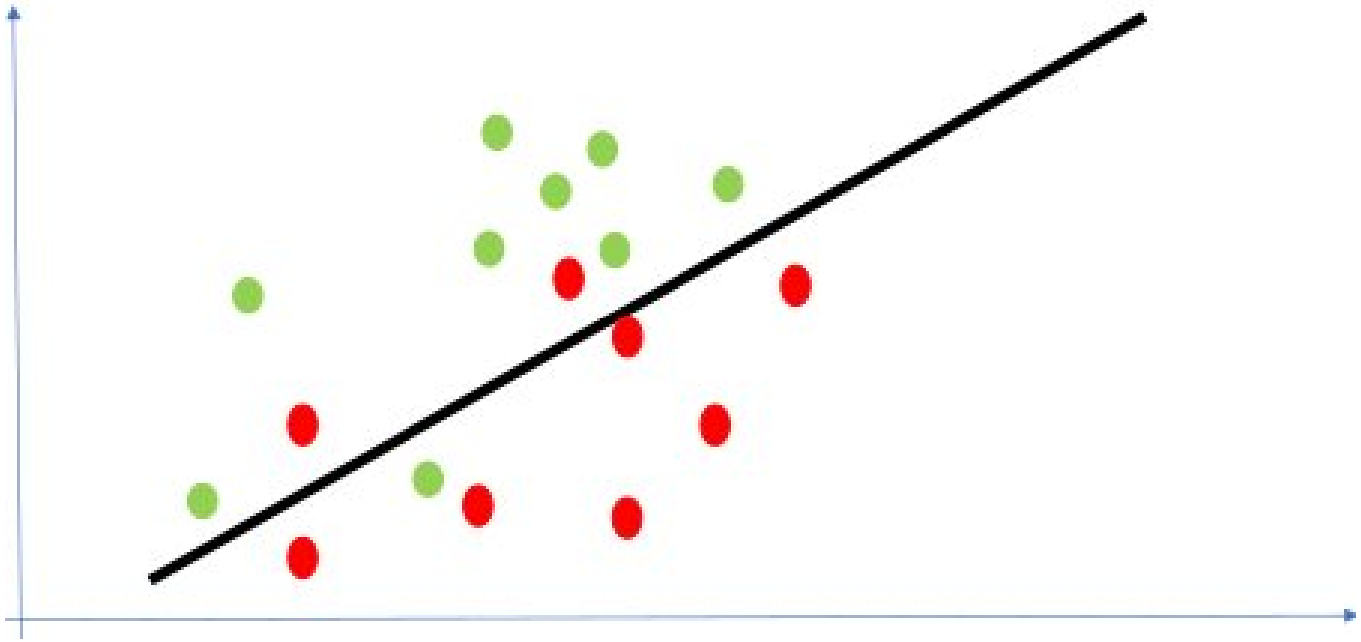
Testing

- How well is my model doing ?
- How do I improve it ?

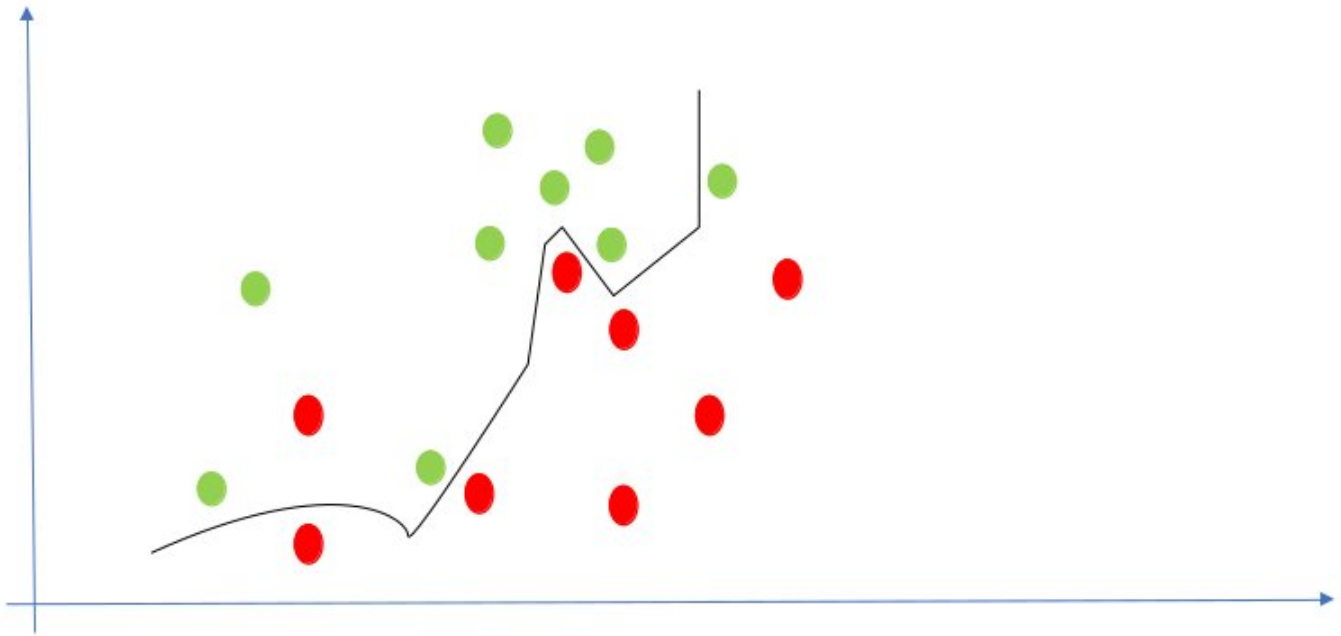
Which model is better ?



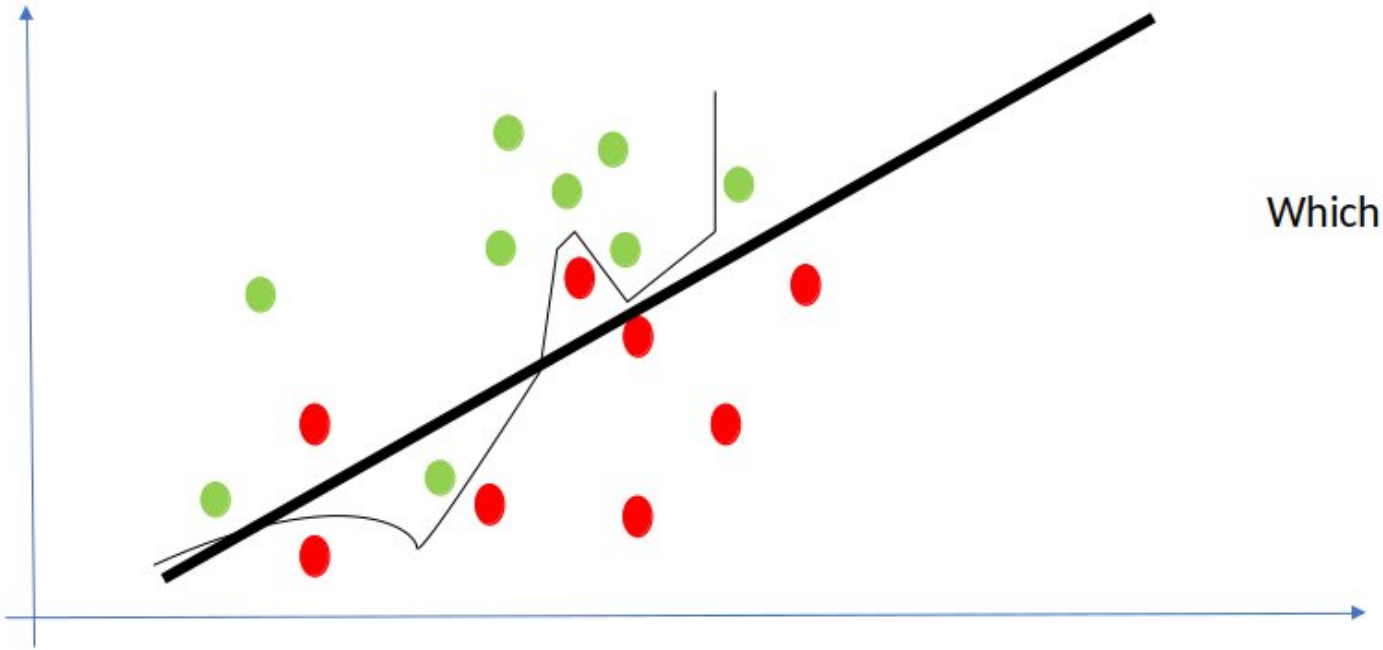
Which model is better ?



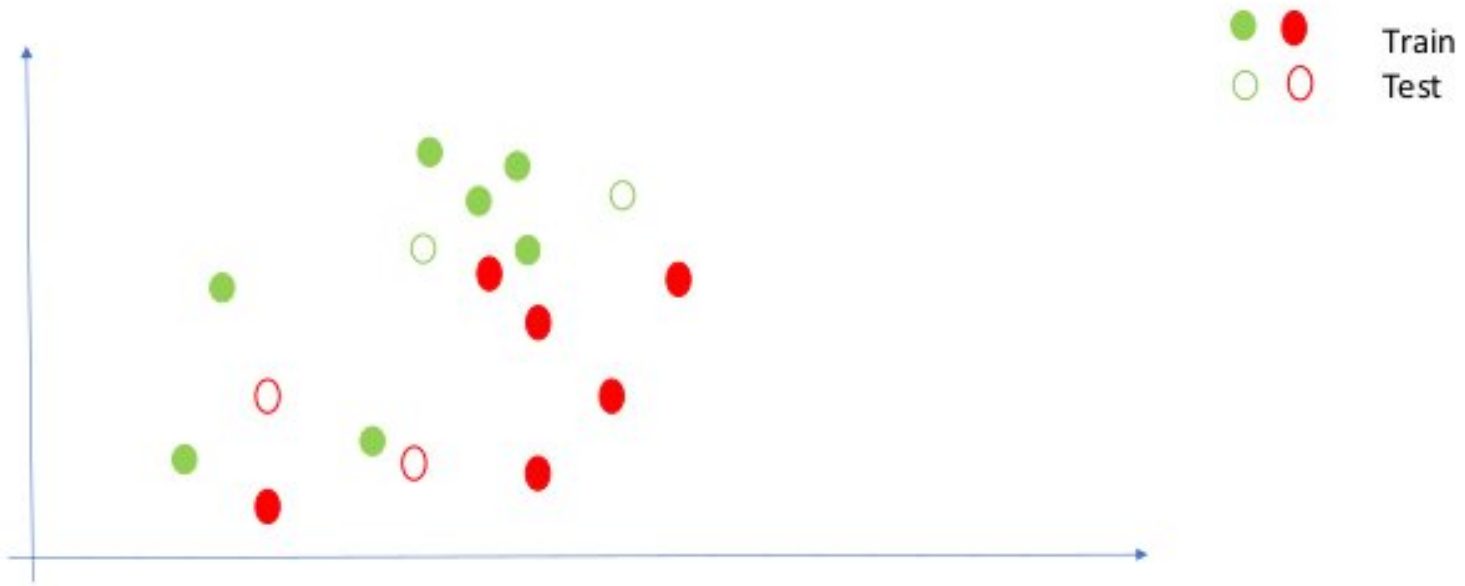
Which model is better ?



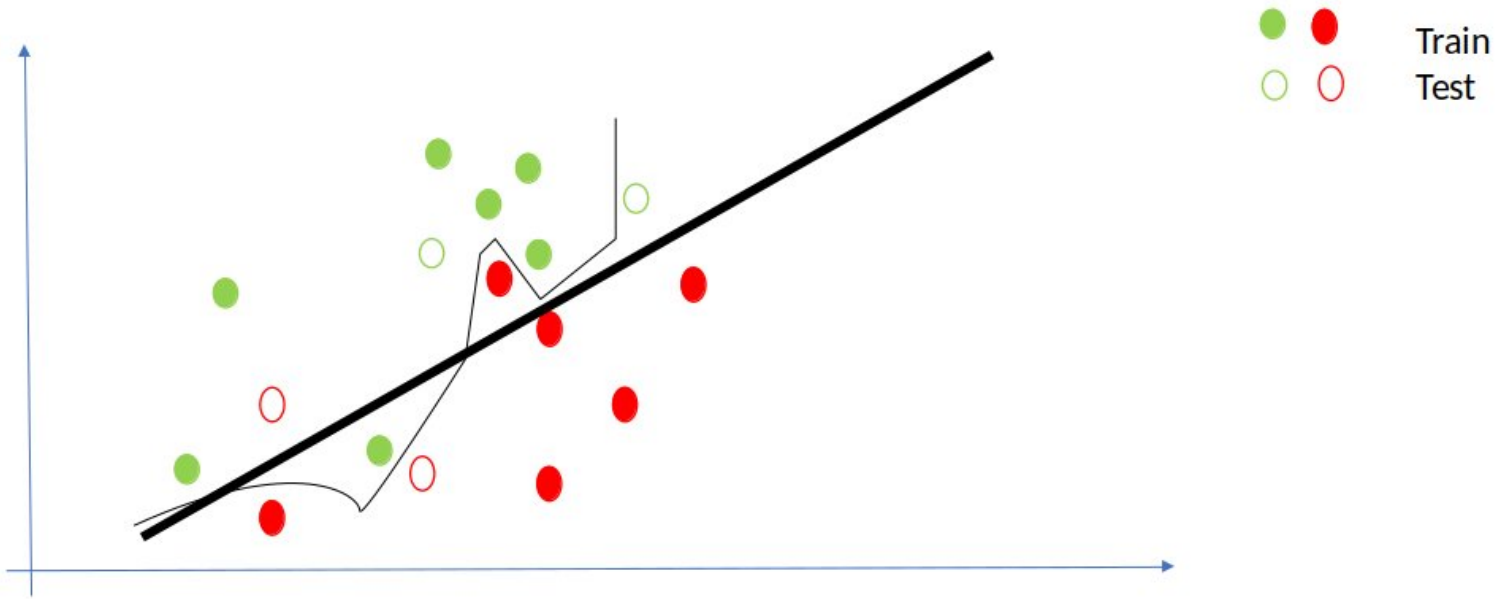
Which model is better ?



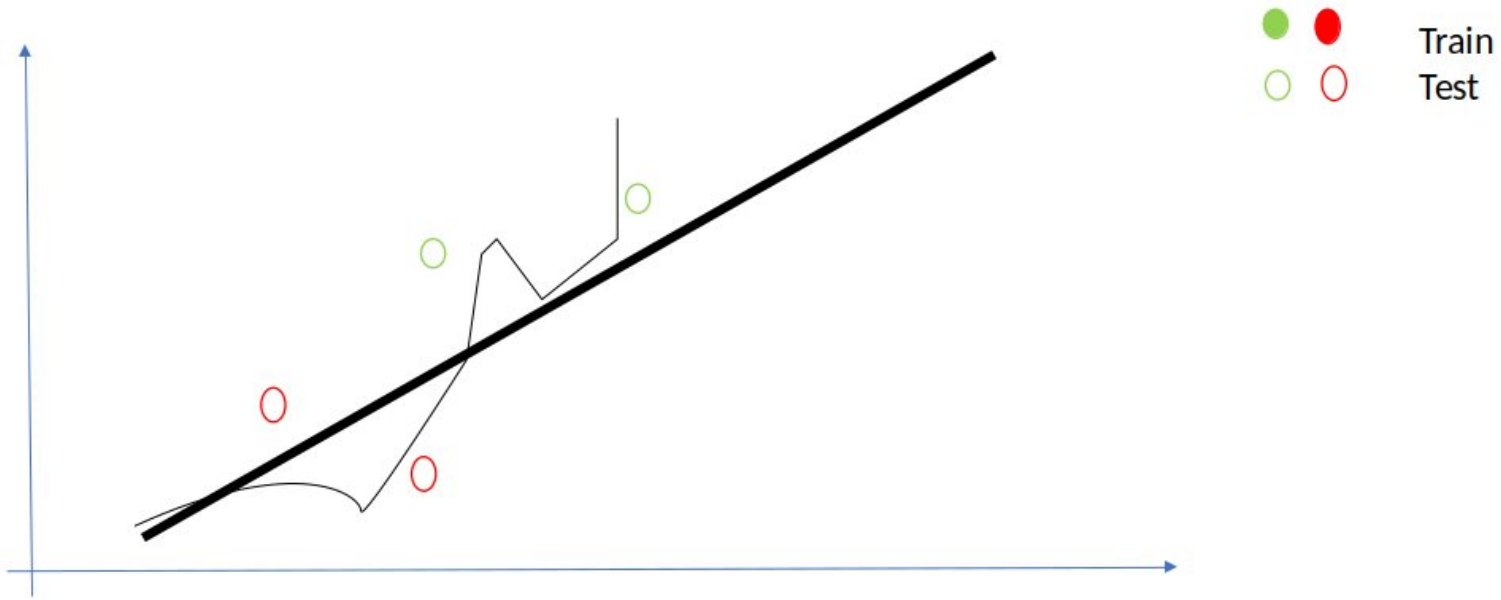
Training vs Testing



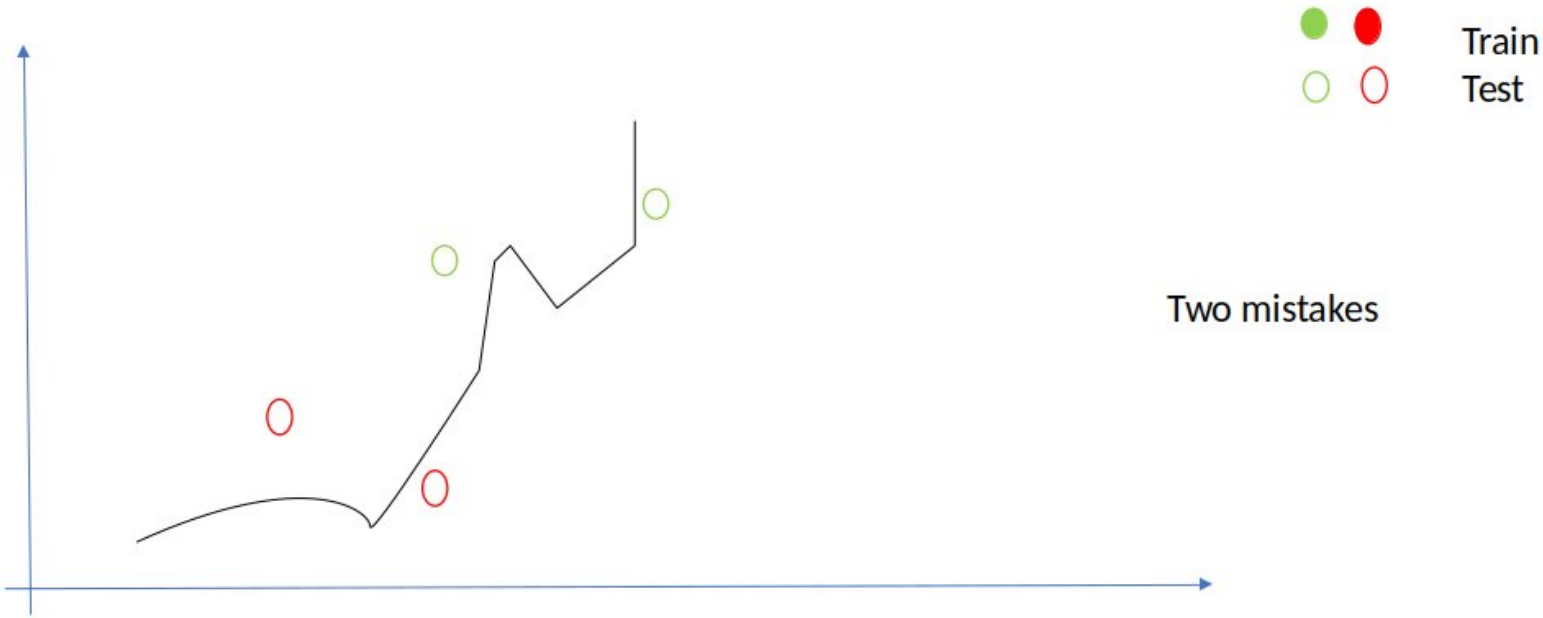
Training vs Testing



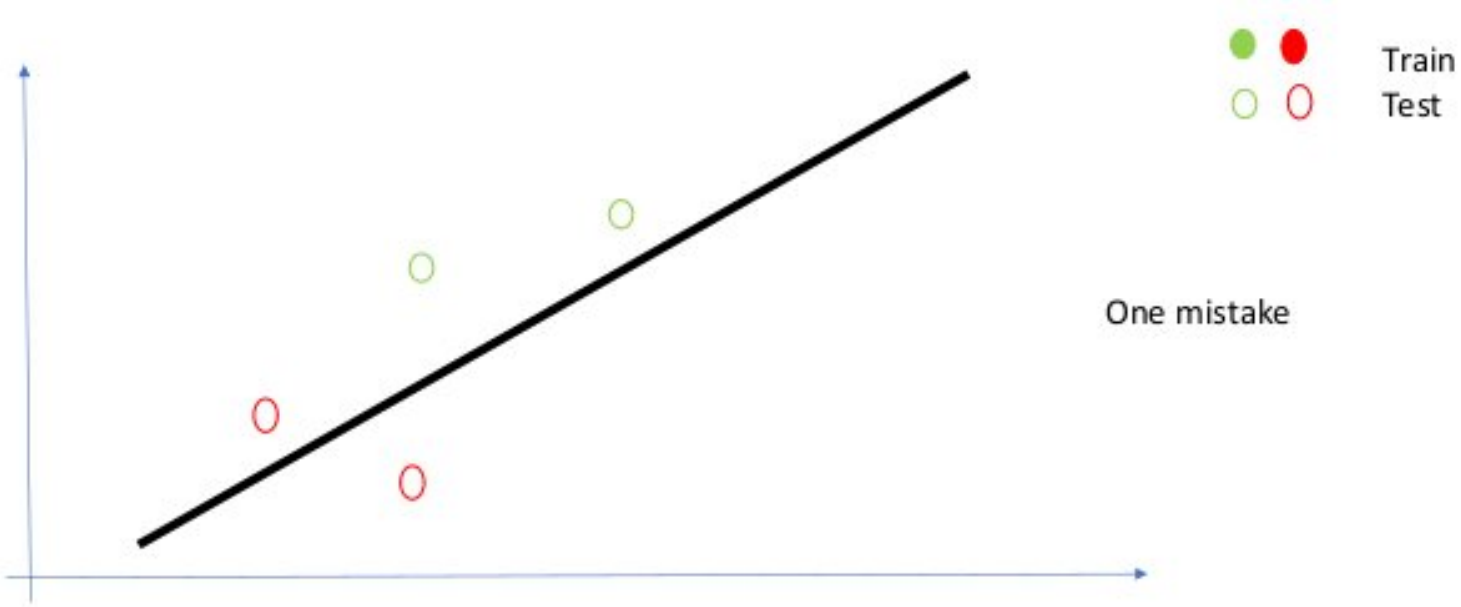
Training vs Testing



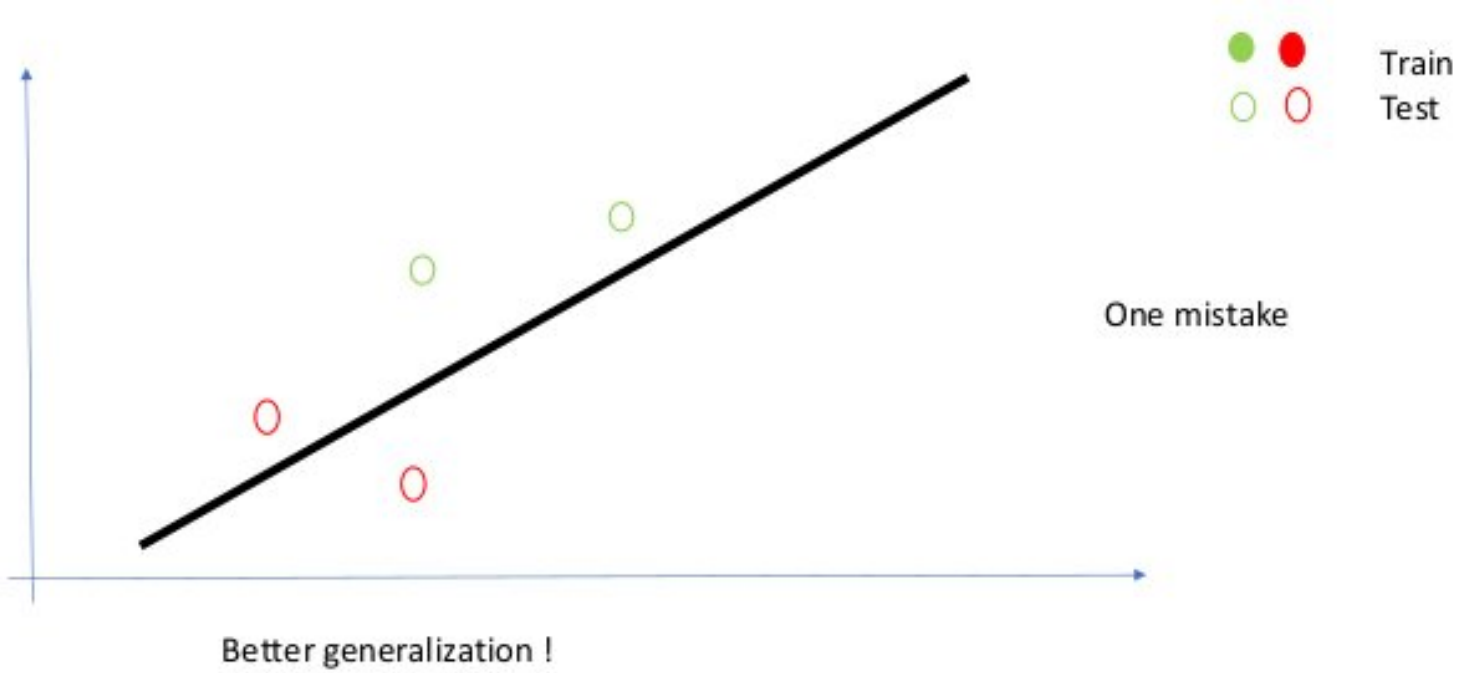
Training vs Testing



Training vs Testing



Training vs Testing



Learning Rule

- NEVER use your testing data for training

Training vs Testing

```
def split_data(data, prob):
  results = [], []
  for row in data:
    results[0 if random.random() < prob else 1].append(row)
  return results

def train_test_split(x, y, test_pct):
  data = zip(x, y)                                # pair corresponding values
  train, test = split_data(data, 1 - test_pct)  # split the data set of pairs
  x_train, y_train = zip(*train)                  # magical un-zip trick
  x_test, y_test = zip(*test)
  return x_train, x_test, y_train, y_test

model = SomeKindOfModel()
x_train, x_test, y_train, y_test = train_test_split(xs, ys, 0.33)
model.train(x_train, y_train)
performance = model.test(x_test, y_test)
```

Learning Rule

- NEVER use your testing data for training

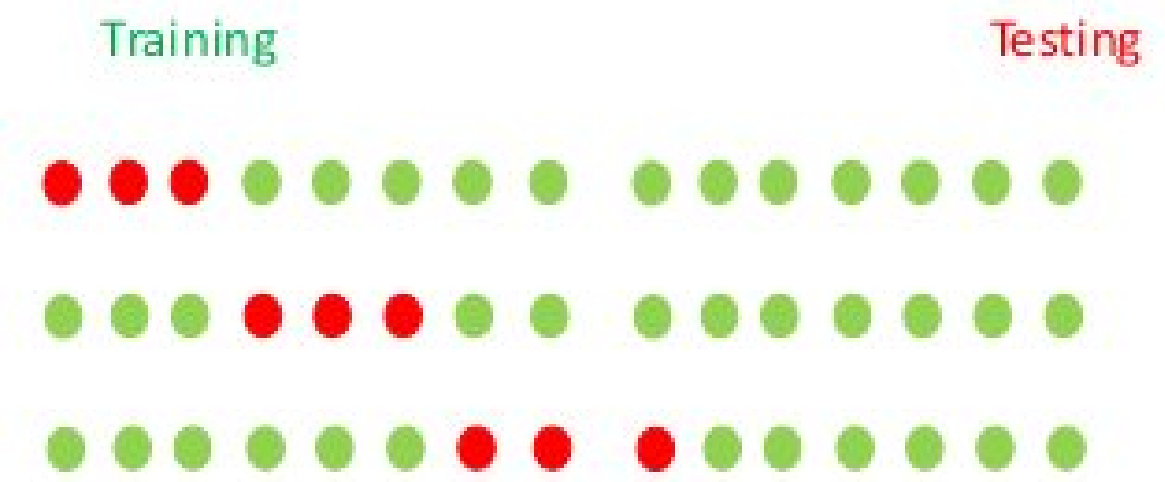
How not losing data ?

K-Fold Cross Validation

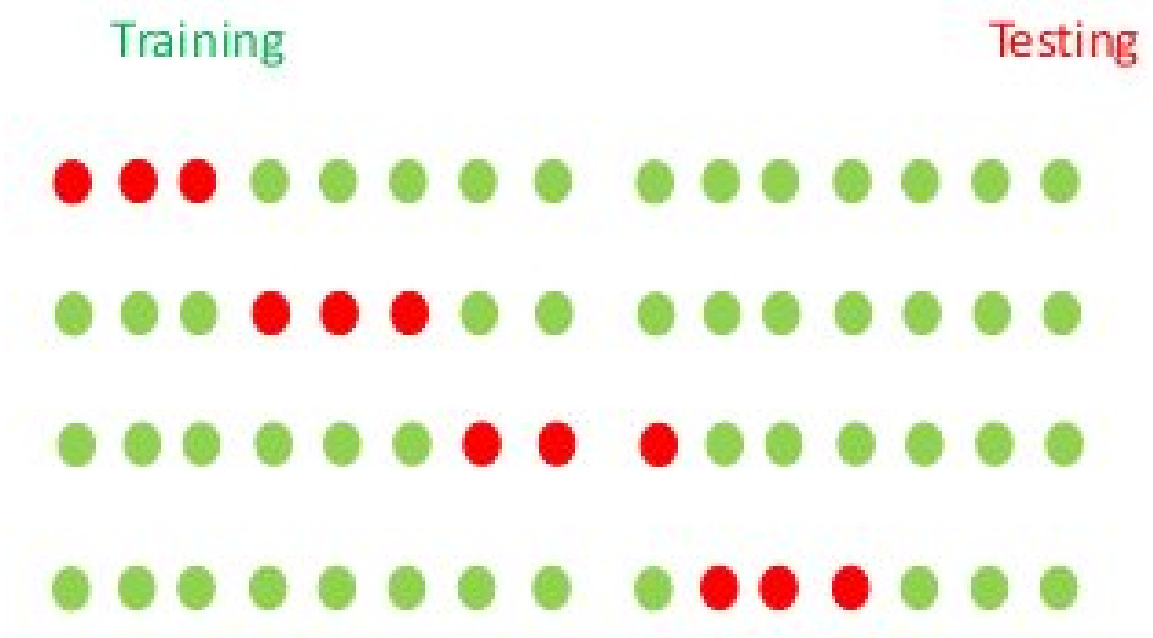
K-Fold Cross Validation

K-Fold Cross Validation

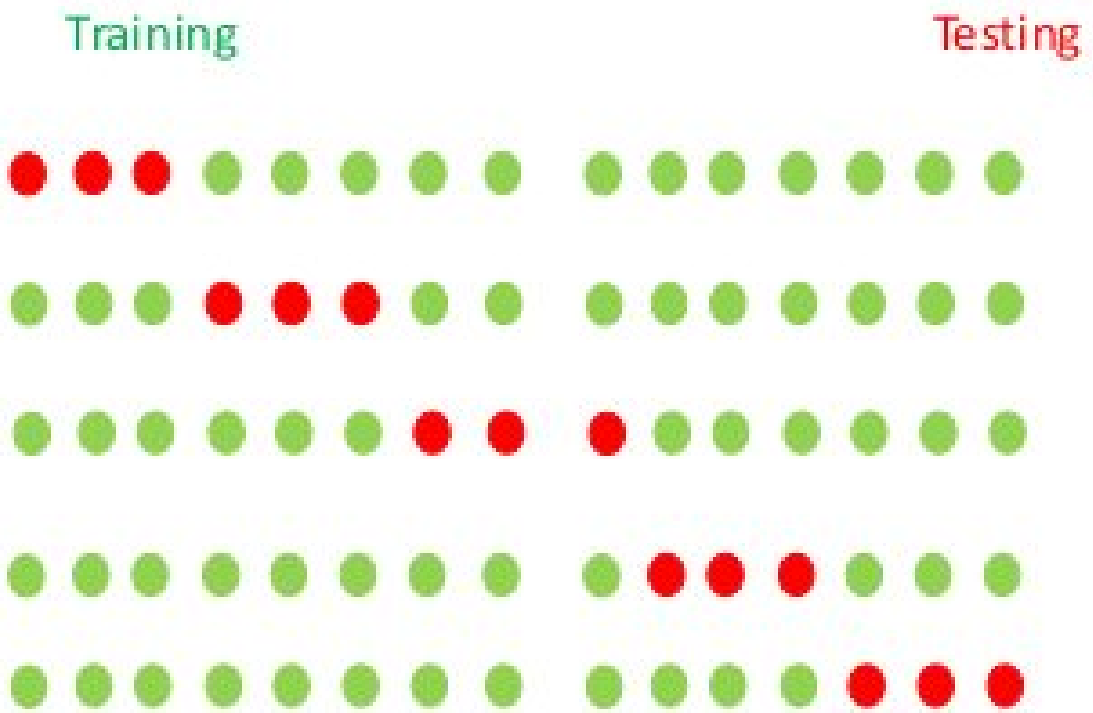
K-Fold Cross Validation



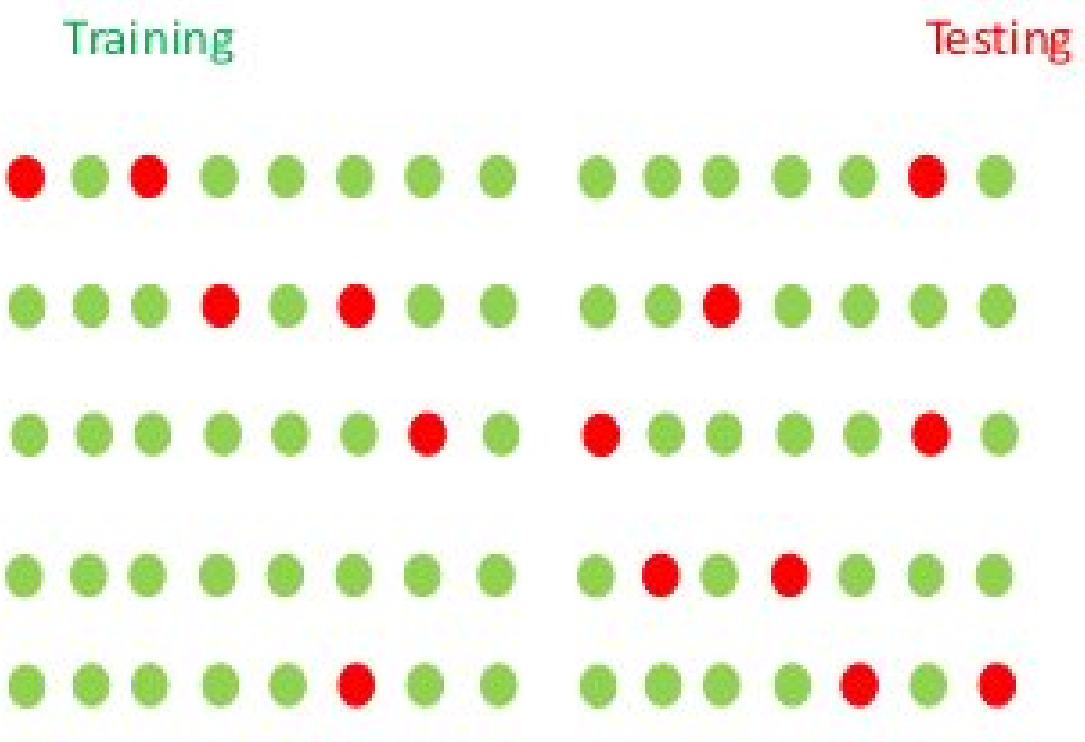
K-Fold Cross Validation



K-Fold Cross Validation



K-Fold Cross Validation



- How well is my model doing ?
  - Tricky question

Example : Credit Card Fraud

Example : Credit Card Fraud

284 335

472

Example : Credit Card Fraud

Model : All transactions are good

Example : Credit Card Fraud

284 335

472

Model : All transactions are good
Correct : 284 335 / (284 335 + 472) = 99.83%

Example : Credit Card Fraud

284 335

472

Model : All transactions are good
Correct : 284 335 / (284 335 + 472) = 99.83%
What about bad transactions ??

Example : Credit Card Fraud

284 335

472

Model : All transactions are fraudulent
Catching all bad transactions
But ...

Example : medical model

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	True Positive 	False Negative 
HEALTHY	False Positive 	True Negative 

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	True Positive 	False Negative 
HEALTHY	False Positive 	True Negative 

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Example : spam model

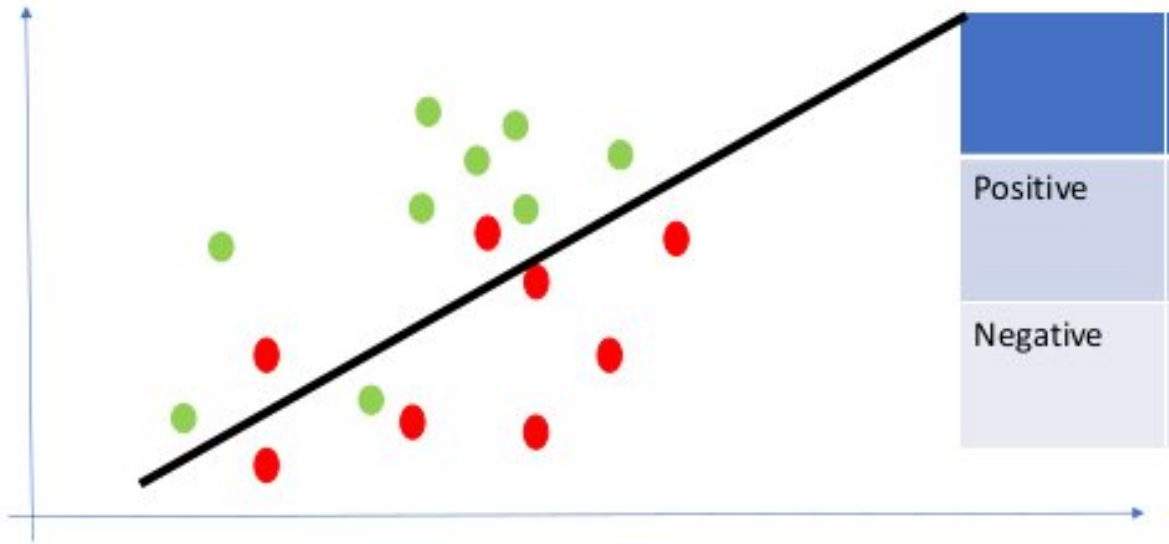
Confusion Matrix

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	True Positive 	False Negative 
NON SPAM	False Positive 	True Negative 

Confusion Matrix

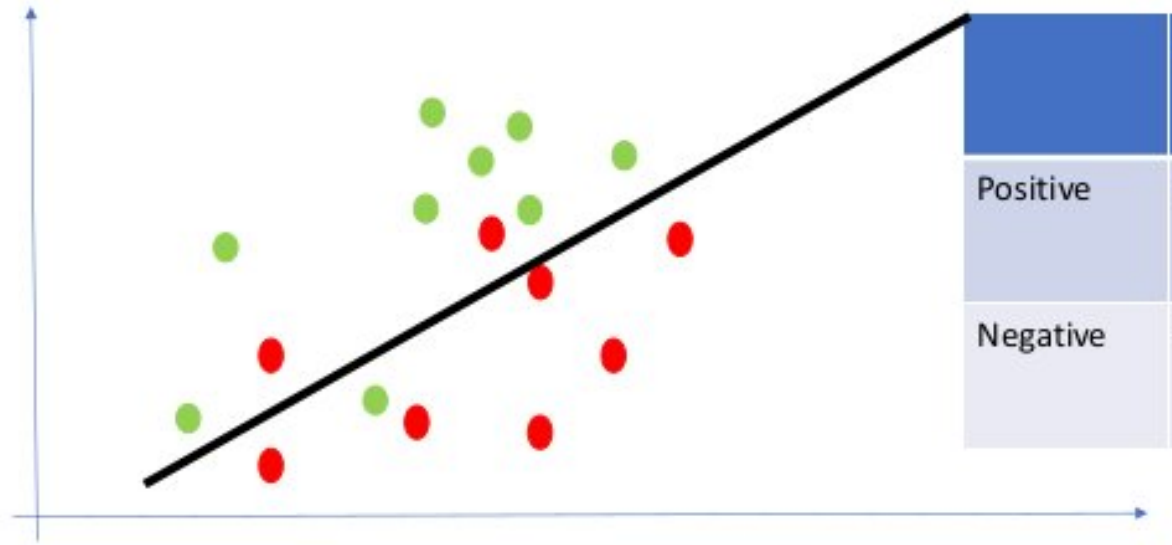
	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Confusion Matrix



	Guessed P	Guessed N
Positive		
Negative		

Confusion Matrix



	Guessed P	Guessed N
Positive	8	1
Negative	2	6

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Accuracy

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Accuracy $= (1000+8000)/10000 = 90\%$

Accuracy

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Accuracy = 80%

Accuracy = Correctly classified / all

Accuracy

```
def accuracy(tp, fp, fn, tn):
  correct = tp + tn
  total = tp + fp + fn + tn
  return correct / total
```

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		False Negative 
HEALTHY	False Positive 	

Confusion Matrix

	Diagnosed SICK	Diagnosed HEALTHY
SICK		 False Negative
HEALTHY	 False Positive	

Precision

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM		False Negative 
NON SPAM	False Positive 	

Precision

- high PRECISION 
- high RECALL 

Precision

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Precision = 1000/ (1000+800) =55,7%

Precision

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Precision= 76.8%

Precision = True Positives/ (True Positives + False Positives)

Precision

```
def precision(tp, fp, fn, tn):
  return tp / (tp + fp)
```

Recall

How many did we classify correctly ?

	Diagnosed SICK	Diagnosed HEALTHY
SICK	1000	200
HEALTHY	800	8000

Recall = 1000/ (1000+200) =83.3%

Recall

How many did we classify correctly ?

	Diagnosed SPAM	Diagnosed NON SPAM
SPAM	100	170
NON SPAM	30	700

Recall= 37%

Recall = True Positives/ (True Positives + False Negatives)

Recall

```
def recall(tp, fp, fn, tn):
  return tp / (tp + fn)
```

Precision and Recall

- Precision : 76,9%
- Recall : 37%

- Precision : 55,7%
- Recall : 83.3%

Average

- Precision : 76,9%
- Recall : 37%
- Average : 56,9%

- Precision : 55,7%
- Recall : 83.3%
- Average : 69,5%

Average not OK

Average


Credit Card
Credit Card
Credit Card
Credit Card
Credit Card
Credit Card

Average

284 335

472

Model : All transactions are good

Precision = 100%
Recall = 0%

Average = 50%

Average

284 335

472

Model : All transactions are fraudulent

Precision = .016%
Recall = 100%

Average = 50%

F1 Score

F1 Score = (2 x Precision x Recall) / (Precision + Recall)

- Precision : 76,9%
- Recall : 37%
- Average : 56,9%
- F1 Score = 50%

- Precision : 55,7%
- Recall : 83.3%
- Average : 69,5%
- F1 Score = (2x55,7x83.3) / (55,7+83,3) = 66%

F1 Score

```
def f1_score(tp, fp, fn, tn):
  p = precision(tp, fp, fn, tn)
  r = recall(tp, fp, fn, tn)
  return 2 * p * r / (p + r)
```

sklearn : Measuring the quality of a predictive algorithm

```
01: # Import datasets, classifiers and performance metrics
02: from sklearn import datasets, svm, metrics
03: digits = datasets.load_digits()

04: # To apply a classifier on this data, we need to flatten the image, to
05: # turn the data in a (samples, feature) matrix:
06: n_samples = len(digits.images)
07: data = digits.images.reshape((n_samples, -1))

08: # Create a classifier: a support vector classifier
09: classifier = svm.SVC(gamma=0.001)

10: # We learn the digits on the first half of the digits
11: classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

12: # Now predict the value of the digit on the second half:
13: expected = digits.target[n_samples // 2:]
14: predicted = classifier.predict(data[n_samples // 2:])
15: print("Classification␣report␣for␣classifier␣%s:\n%s\n" \
16:     % (classifier, metrics.classification_report(expected, predicted)))
17: print("Confusion␣matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
```

sklearn : Measuring the quality of a predictive algorithm

```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```

sklearn : Measuring the quality of a predictive algorithm

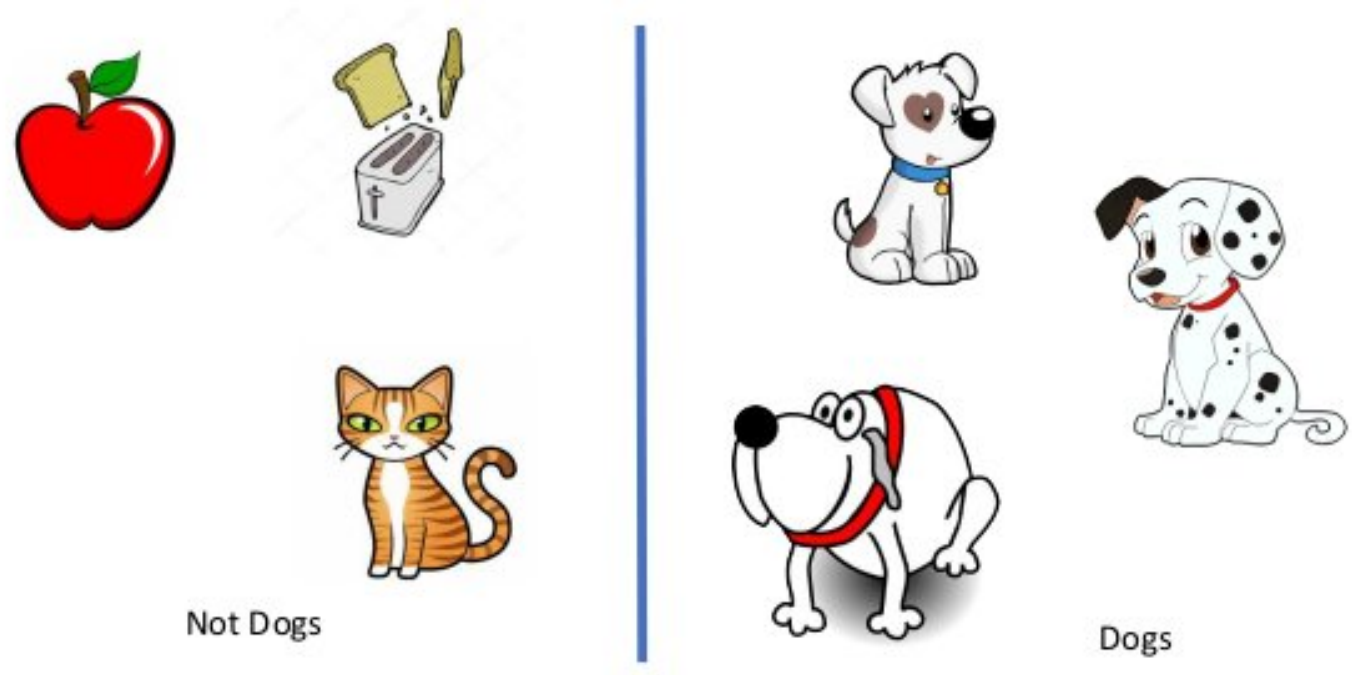
```
Confusion matrix:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

Underfitting/overfitting

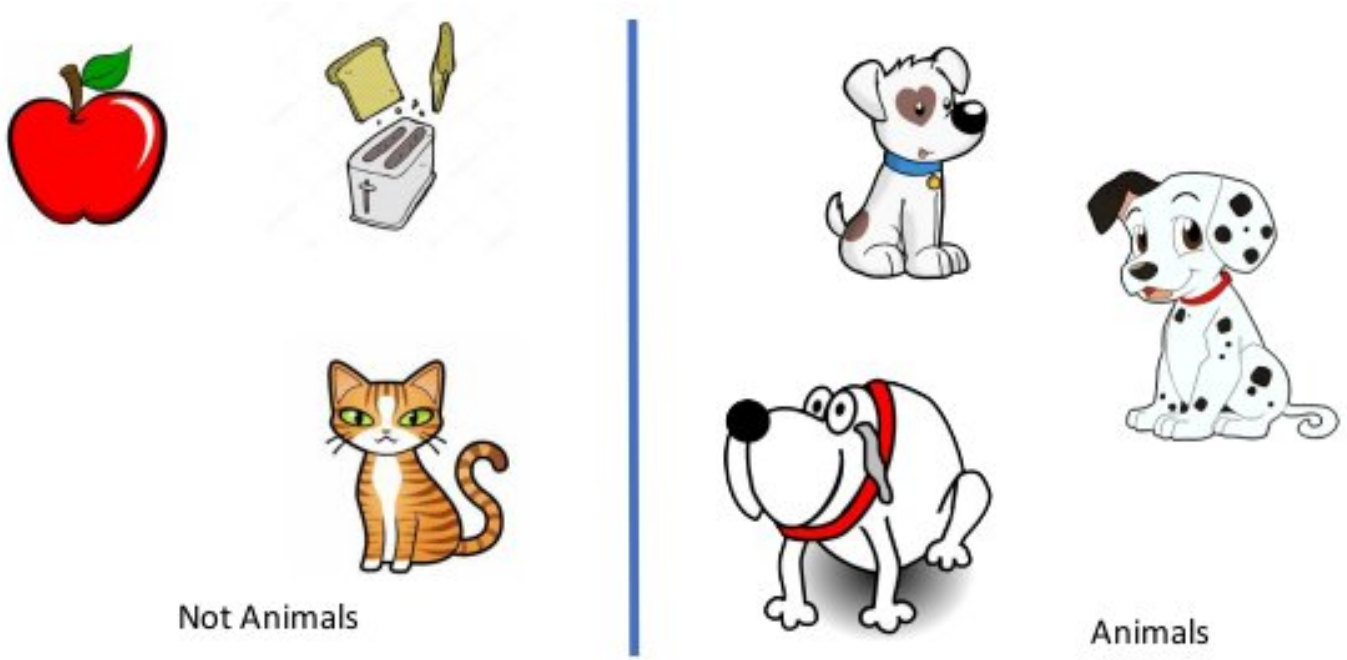




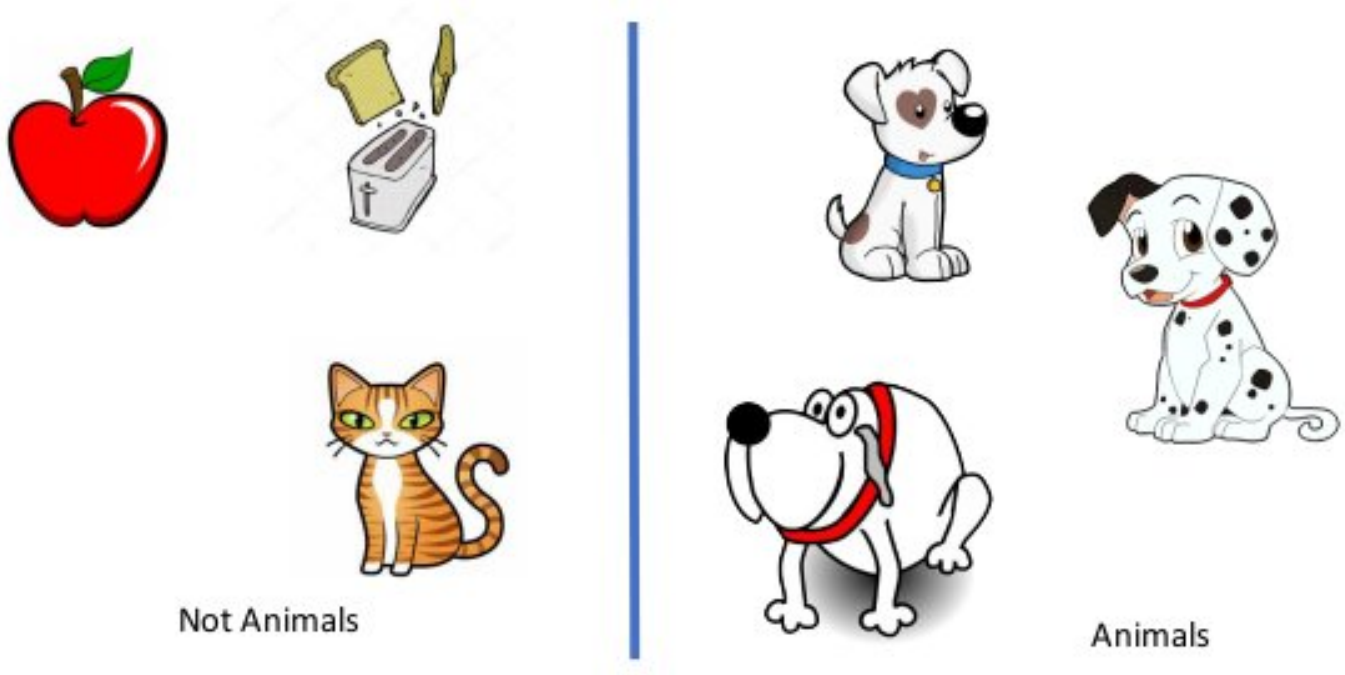

Underfitting/overfitting



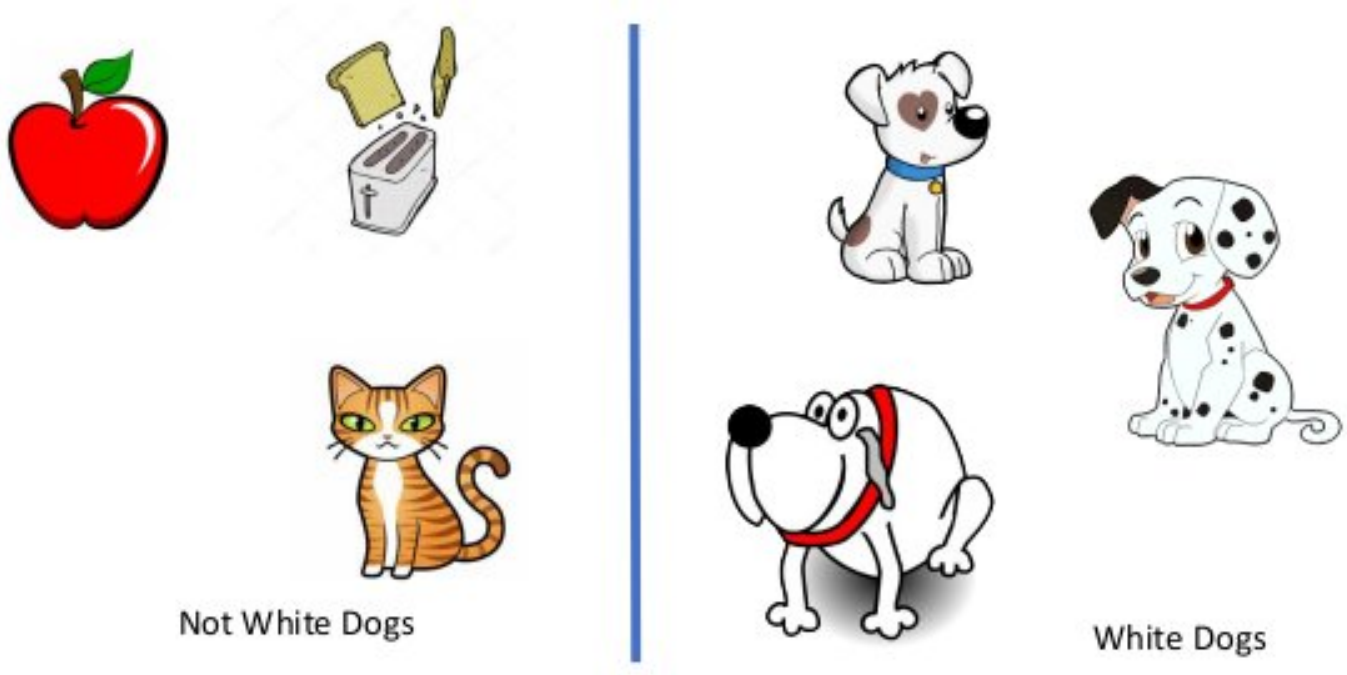
Underfitting/overfitting



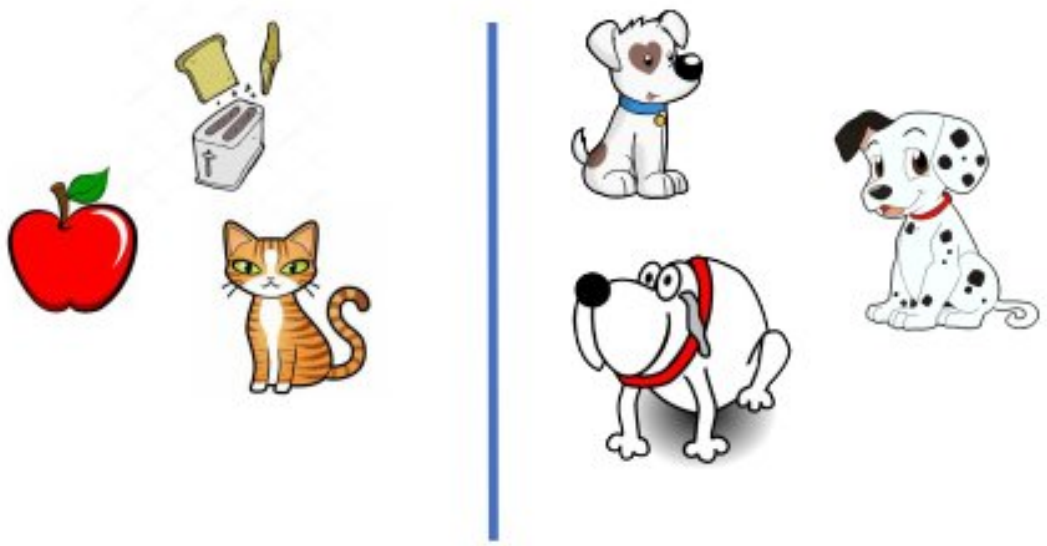
Underfitting/overfitting



Underfitting/overfitting



Underfitting/overfitting



Underfitting : Not Animals
Overfitting : Not White Dogs
OK : Not Dogs

Underfitting : Animals
Overfitting : White Dogs
OK : Dogs

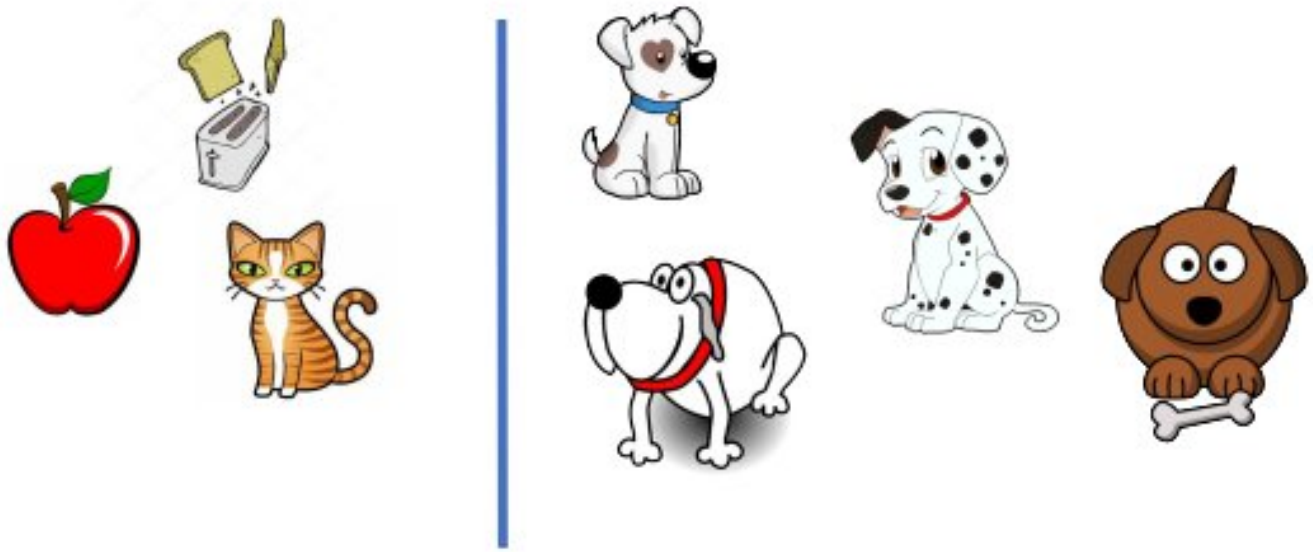
Underfitting/overfitting

Training set :
Bad
Great
Good

Underfitting : Not Animals
Overfitting : Not White Dogs
OK : Not Dogs

Underfitting : Animals
Overfitting : White Dogs
OK : Dogs

Underfitting/overfitting



Testing set :

Bad	Underfitting : Not Animals	Underfitting : Animals
Bad	Overfitting : Not White Dogs	Overfitting : White Dogs
Good	OK : Not Dogs	OK : Dogs

the more data you have, the harder it is to over- fit.

To sum up

- Define problem (data)
- List tools (algorithms)
- Evaluate tools to find the best one
  - Accuracy
  - Precision
  - Recall
  - F1

Features

- As we mentioned, when your data doesn't have enough features, your model is likely to underfit.
- When your data has too many features, it's easy to overfit.
- What are features and where do they come from ?

Features are whatever inputs we provide to our model.

Type of features

Type of features we have constrains the type of models we can use :

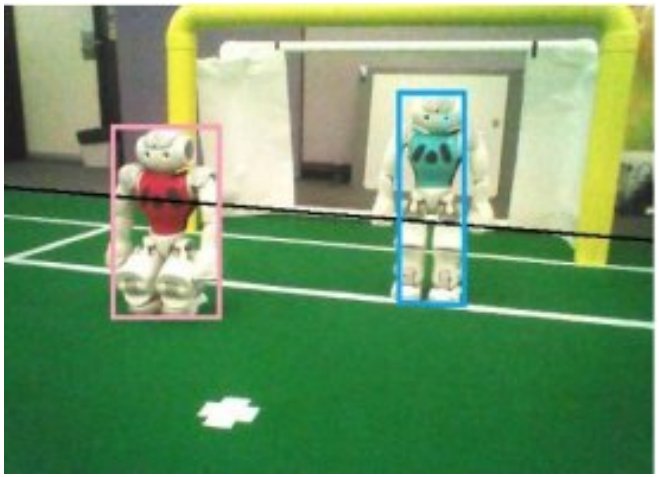
- The Naive Bayes classifier is suited to *yes-or-no features*
- Regression models require *numeric features*
- Decision trees can deal with *numeric or categorical data*.

Features extraction

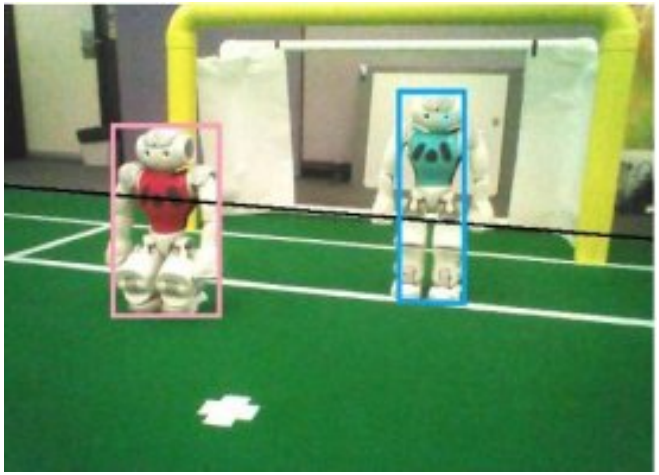
- feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps
- Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set

Example : robot detection

Example : robot detection

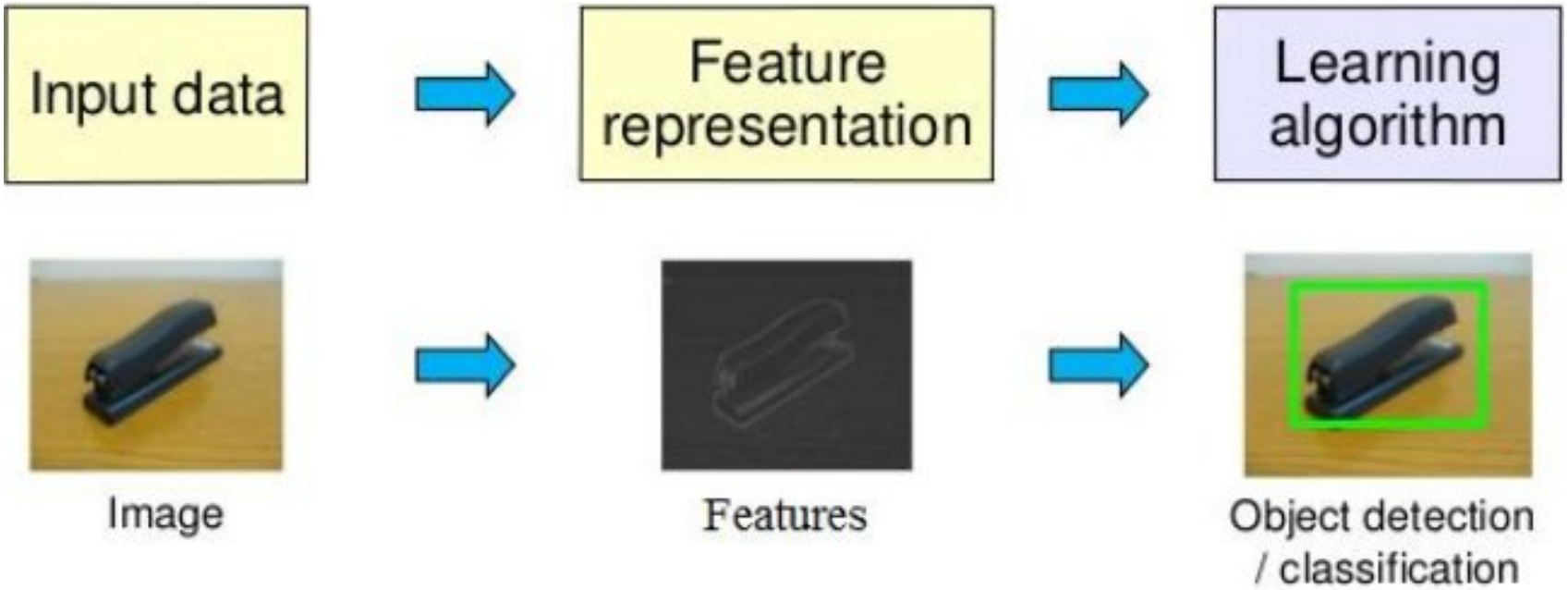


Example : robot detection



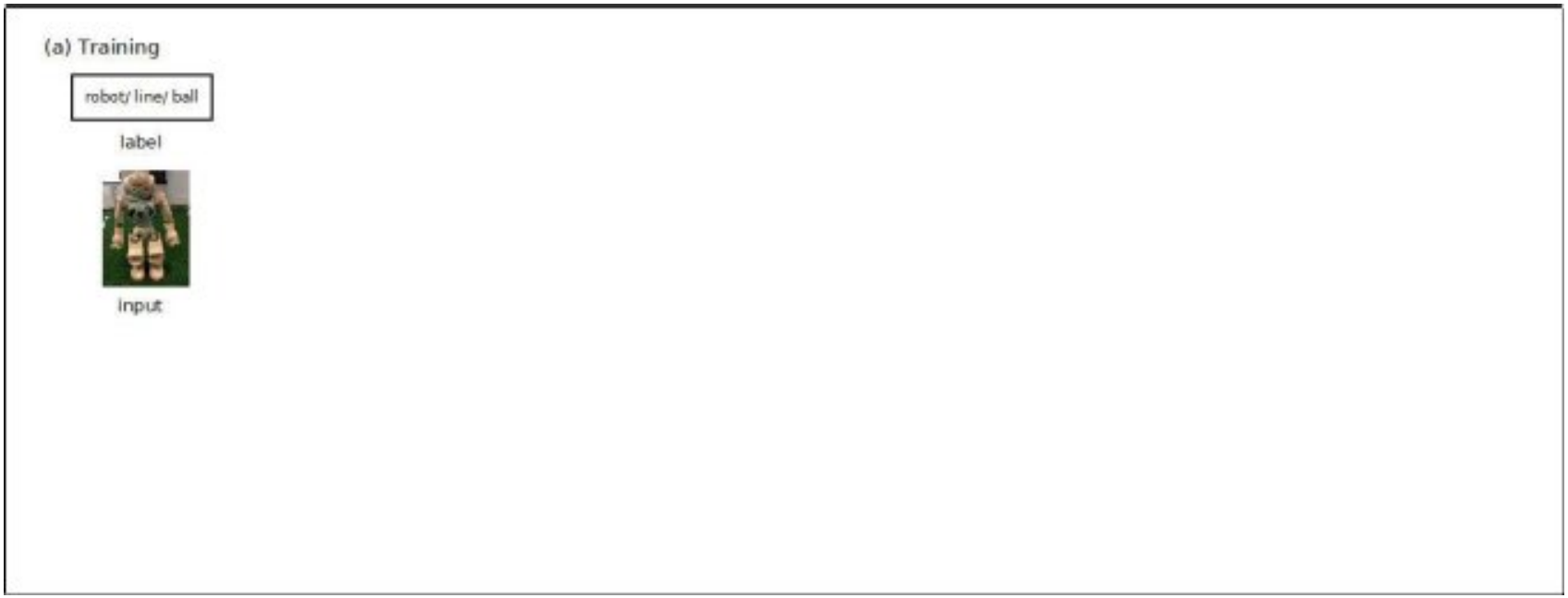
Can we detect robot using low quality images ?

Example : robot detection

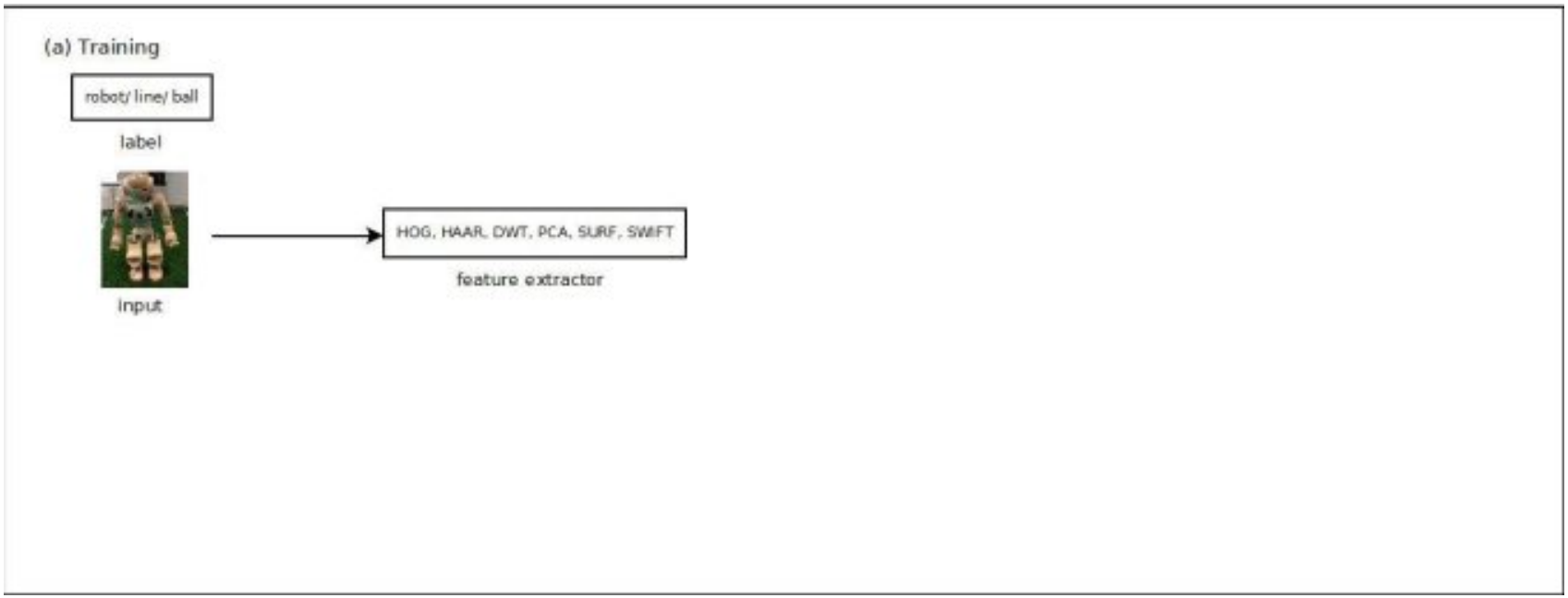


Example : robot detection

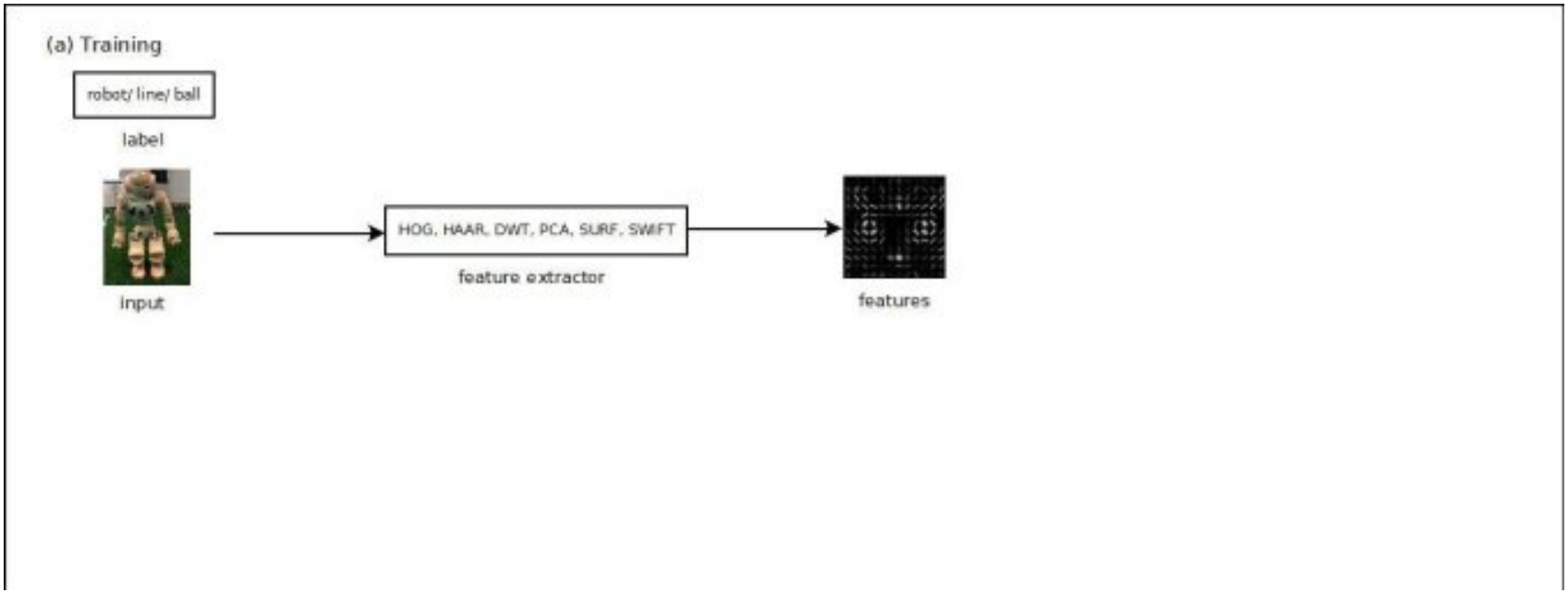
Example : robot detection



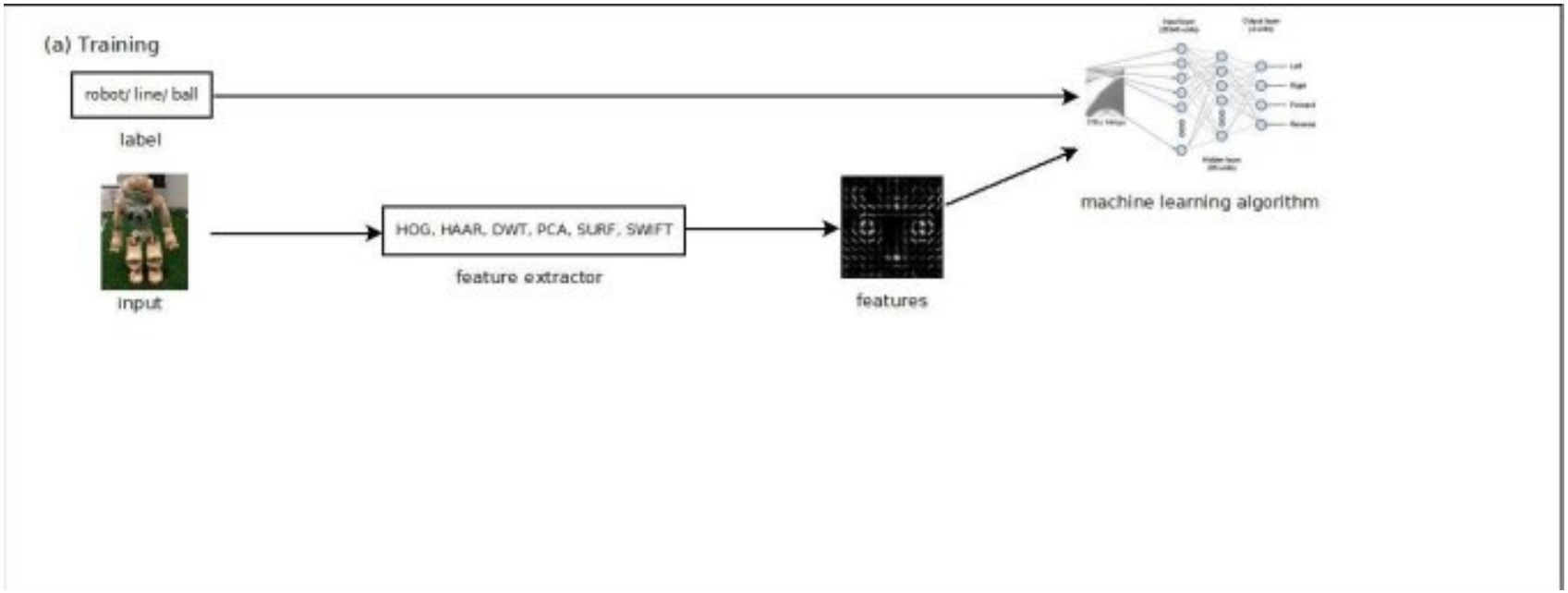
Example : robot detection



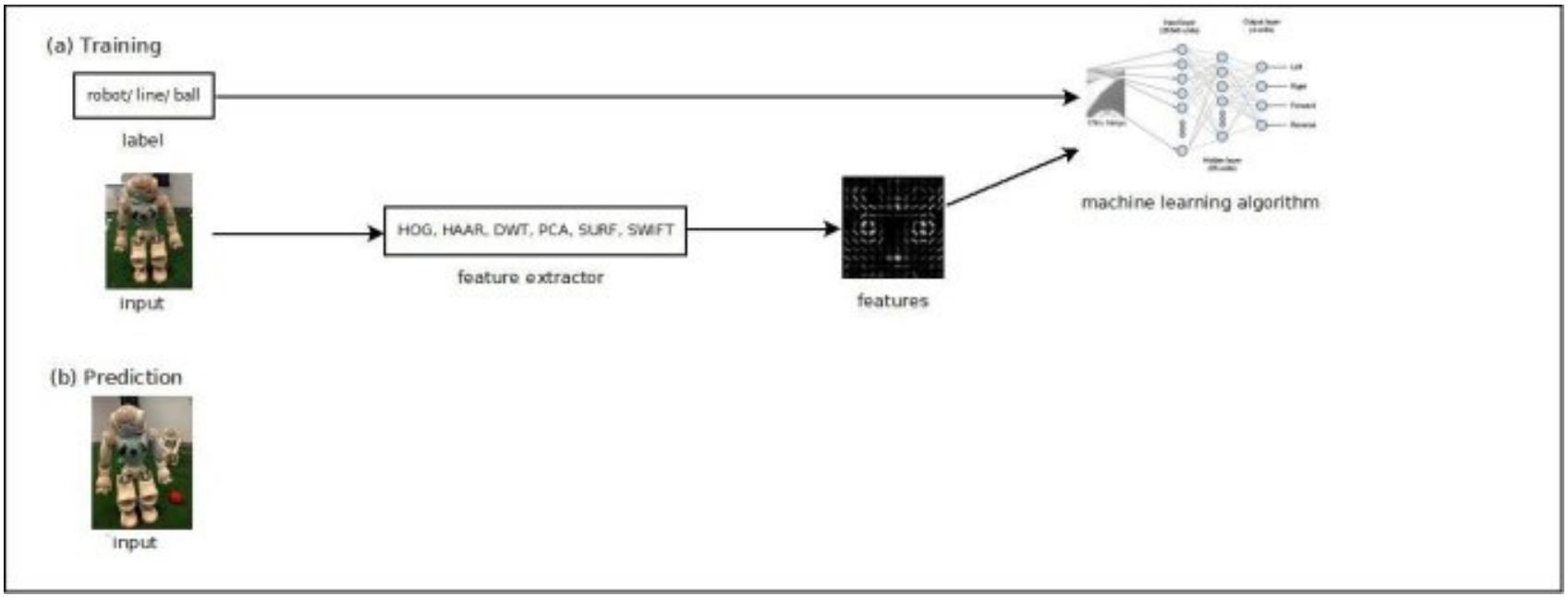
Example : robot detection



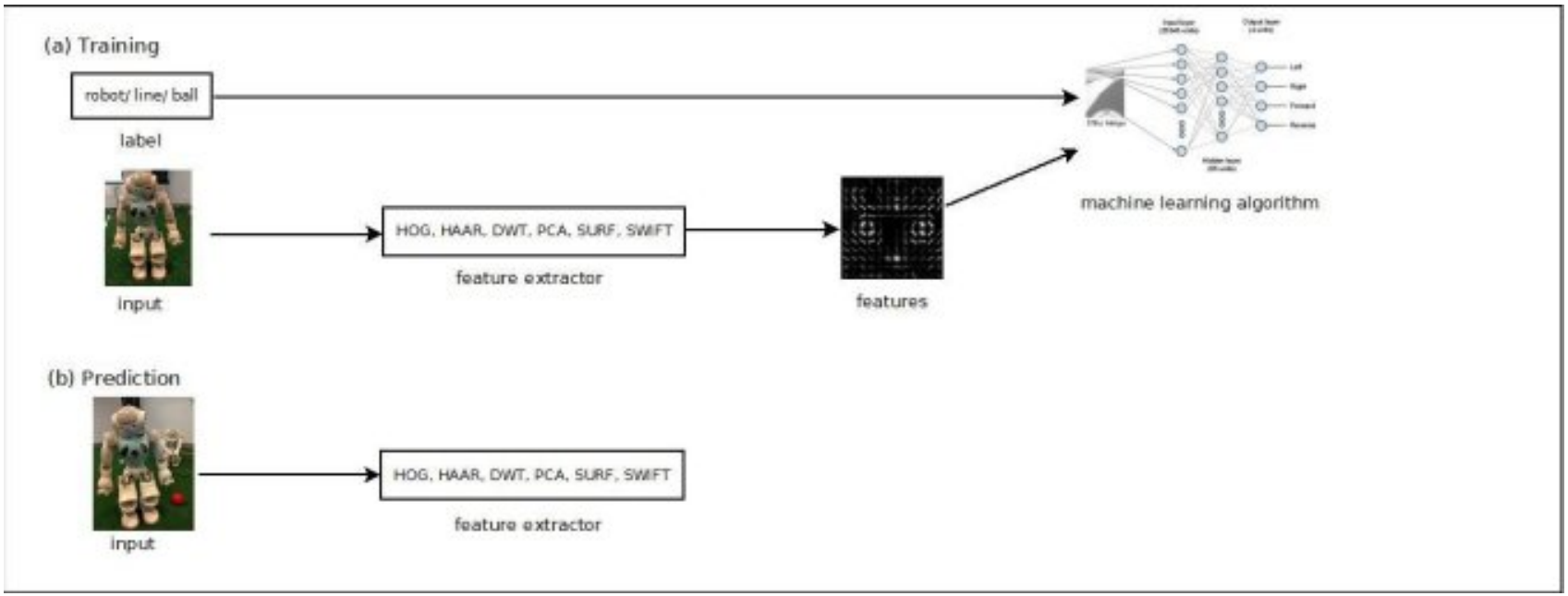
Example : robot detection



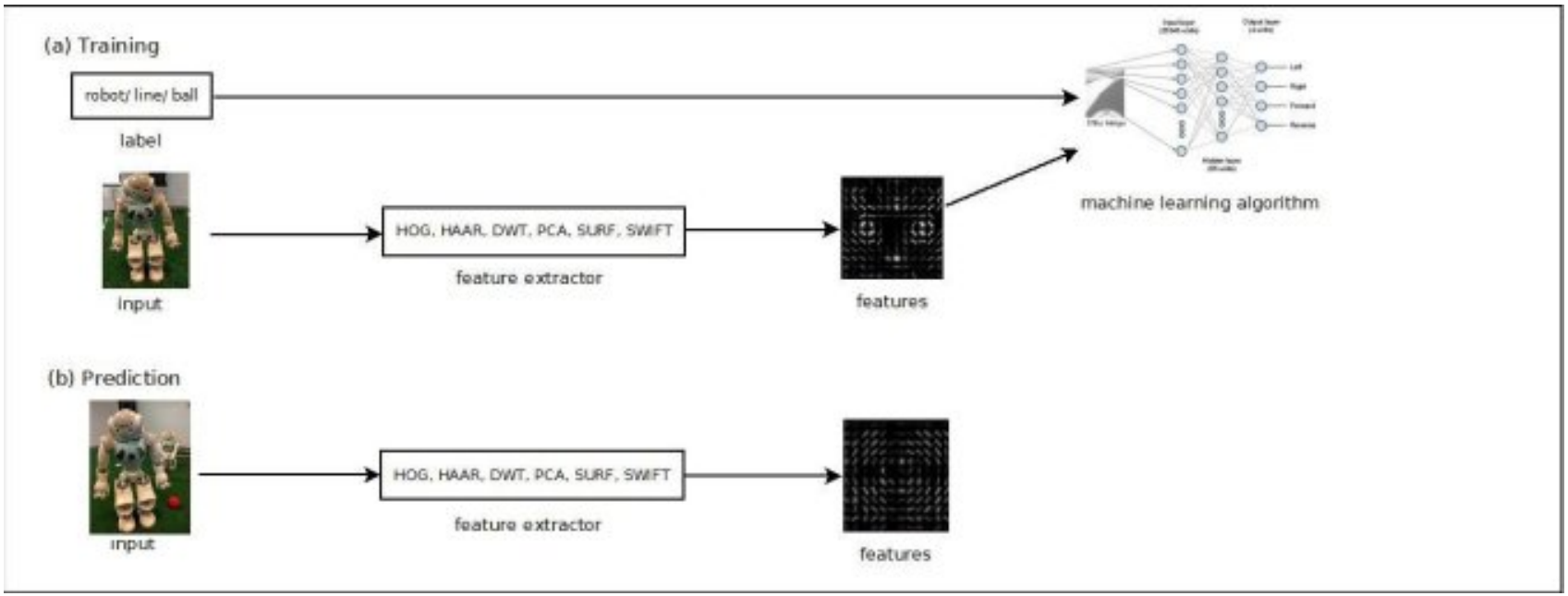
Example : robot detection



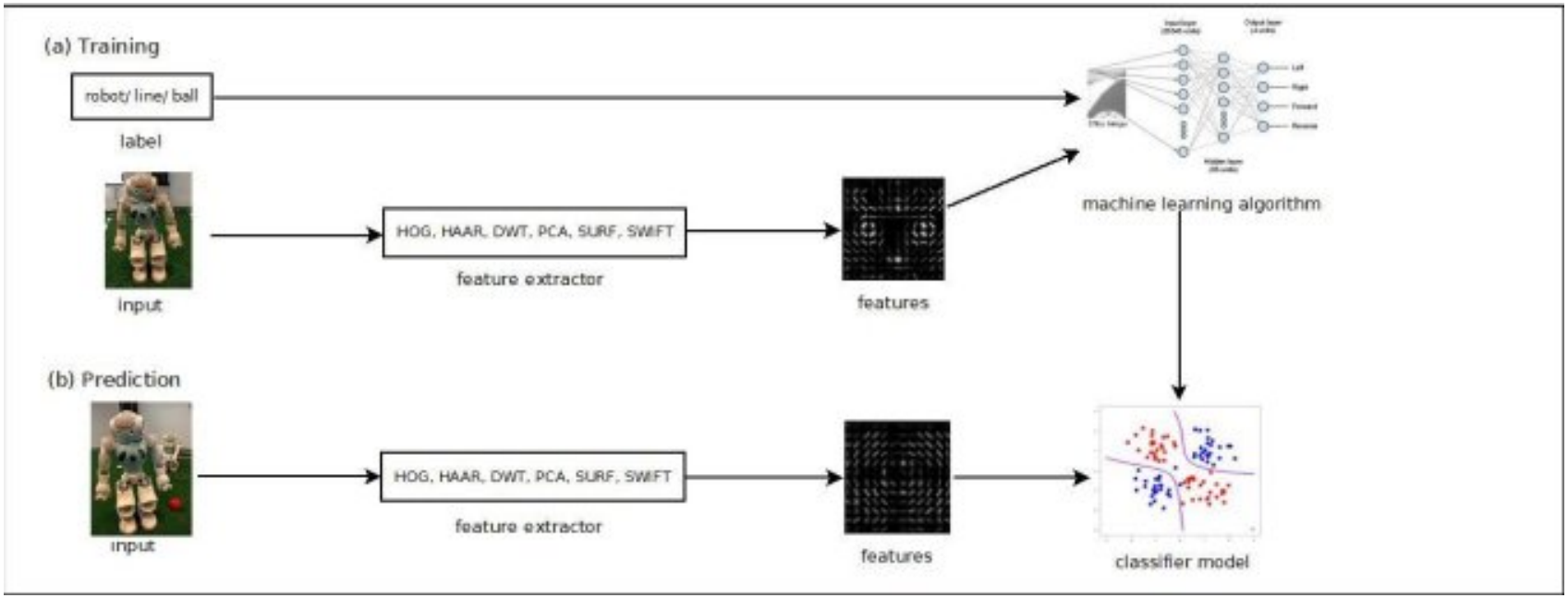
Example : robot detection



Example : robot detection



Example : robot detection

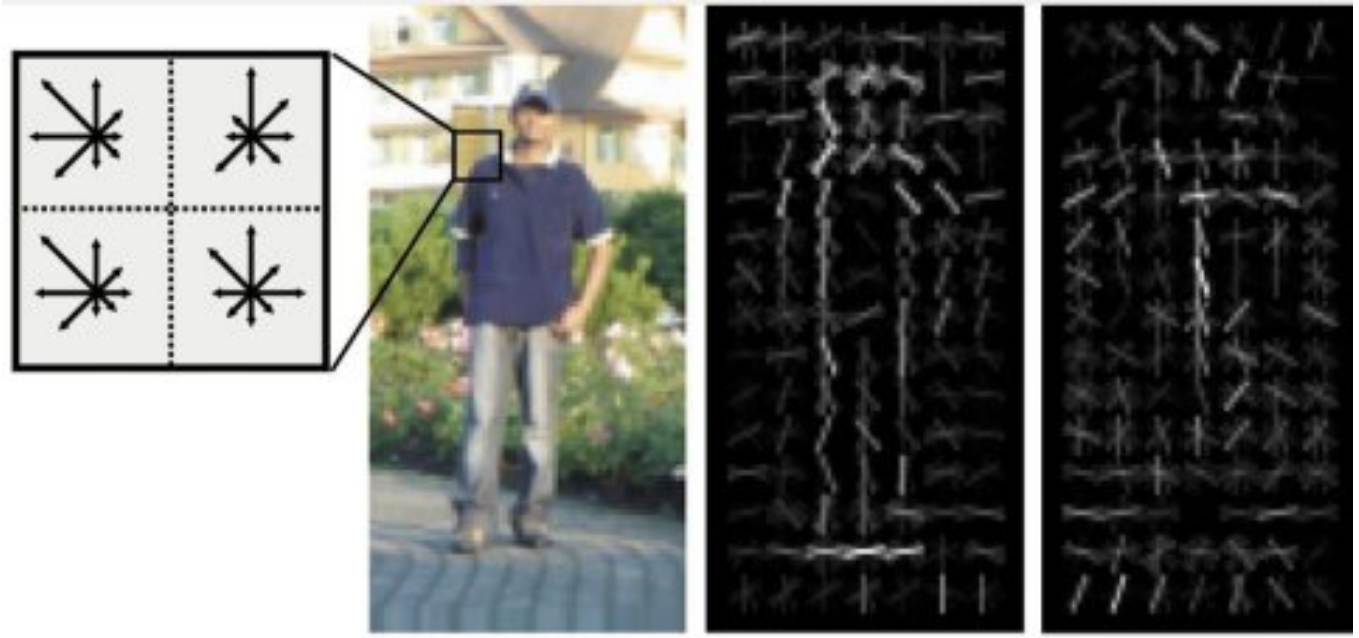


HOG+SVM

- Application : Persons detector
- HOG : Histograms of Oriented Gradients
- The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. This makes the classification task easier.
- The creators of this approach trained a Support Vector Machine (a type of machine learning algorithm for classification), or "SVM", to recognize HOG descriptors of people.

HOG+SVM

HOG : entire person is represented by a single feature vector



HOG+SVM : sliding detection

The HOG person detector uses a sliding detection window which is moved around the image.

```
# import the necessary packages
import imutils
import argparse
import time
import cv2

def pyramid(image, scale=1.5, minSize=(30, 30)):
        # yield the original image
        yield image

        # keep looping over the pyramid
        while True:
                # compute the new dimensions of the image and resize it
                w = int(image.shape[1] / scale)
                image = imutils.resize(image, width=w)

                # if the resized image does not meet the supplied minimum
                # size, then stop constructing the pyramid
                if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:
                        break

                # yield the next image in the pyramid
                yield image
...
```

HOG+SVM : sliding detection

```
...
def sliding_window(image, stepSize, windowSize):
        # slide a window across the image
        for y in range(0, image.shape[0], stepSize):
                for x in range(0, image.shape[1], stepSize):
                        # yield the current window
                        yield (x, y, image[y:y + windowSize[1], x:x + windowSize[0]])
```

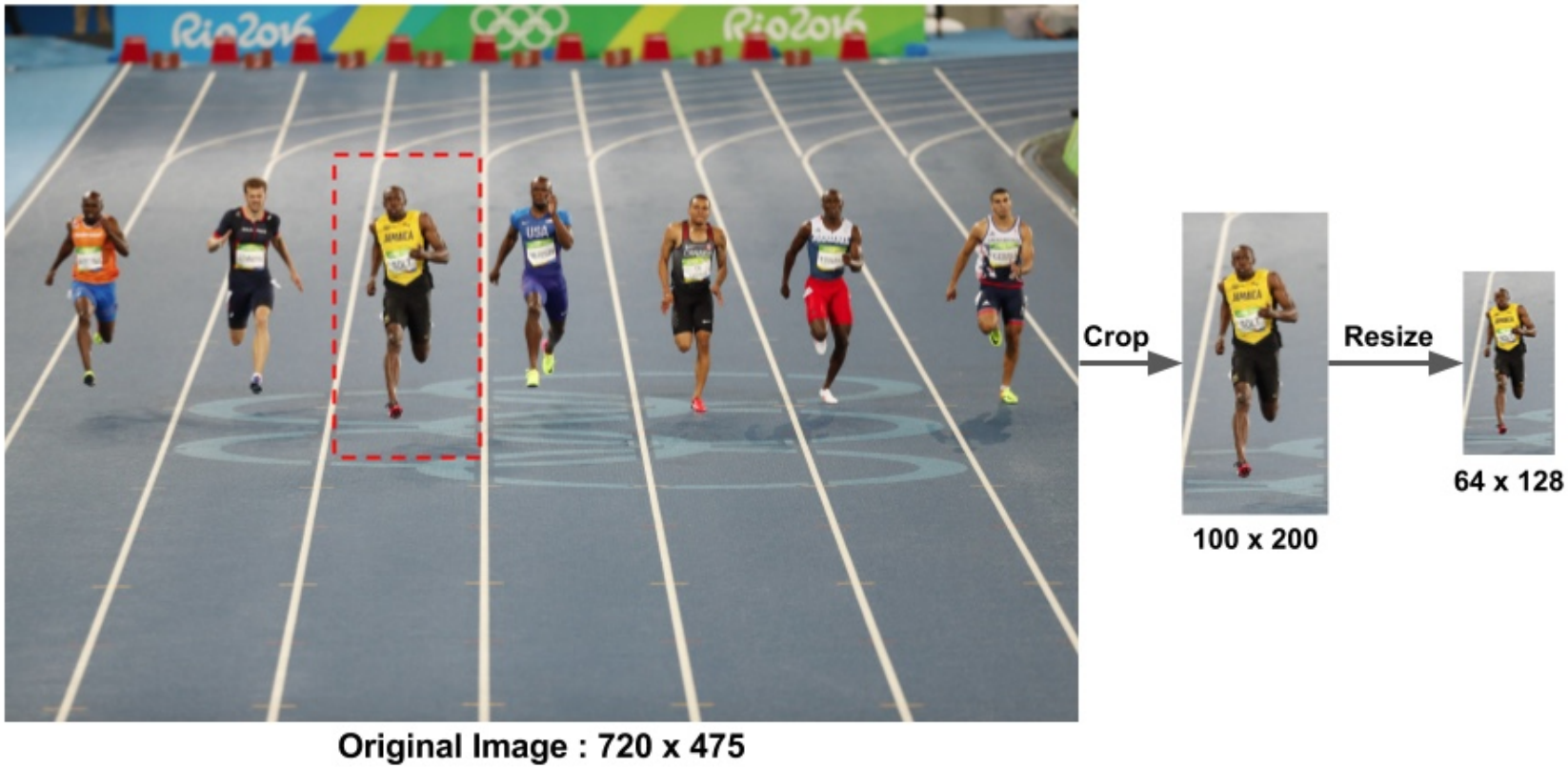
HOG+SVM : sliding detection

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True, help="Path to the image")
args = vars(ap.parse_args())

# load the image and define the window width and height
image = cv2.imread(args["image"])
(winW, winH) = (128, 128)

# loop over the image pyramid
for resized in pyramid(image, scale=1.5):
        # loop over the sliding window for each layer of the pyramid
        for (x, y, window) in sliding_window(resized, stepSize=32, windowSize=(winW, winH)):
                # if the window does not meet our desired window size, ignore it
                if window.shape[0] != winH or window.shape[1] != winW:
                        continue

                # WHERE APPLY A CLASSIFIER

                # since we do not have a classifier, we will just draw the window
                clone = resized.copy()
                cv2.rectangle(clone, (x, y), (x + winW, y + winH), (0, 255, 0), 2)
                cv2.imshow("Window", clone)
                cv2.waitKey(1)
                time.sleep(0.025)
```

HOG+SVM

- At each position of the detector window, a HOG descriptor is computed for the detection window.
- This descriptor is then shown to the trained SVM, which classifies it as either “person” or “not a person”.
- To recognize persons at different scales, the image is subsampled to multiple sizes. Each of these subsampled images is searched

HOG : Step 1 : Preprocessing



HOG : Step 2 : Calculate the Gradient Images

-1	0	1

-1
0
1

HOG : Step 2 : Calculate the Gradient Images

```
# Read image
im = cv2.imread('bolt.png')
im = np.float32(im) / 255.0

# Calculate gradient
gx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=1)
gy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=1)

# Python Calculate gradient magnitude and direction ( in degrees )
mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
```

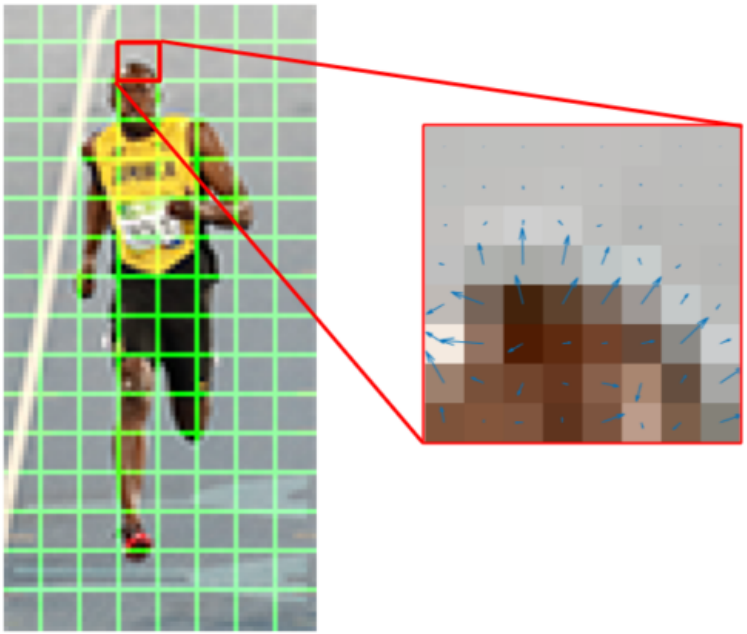
HOG : Step 2 : Calculate the Gradient Images



Left : Absolute value of x-gradient.
Center : Absolute value of y-gradient.
Right : Magnitude of gradient.

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells

HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

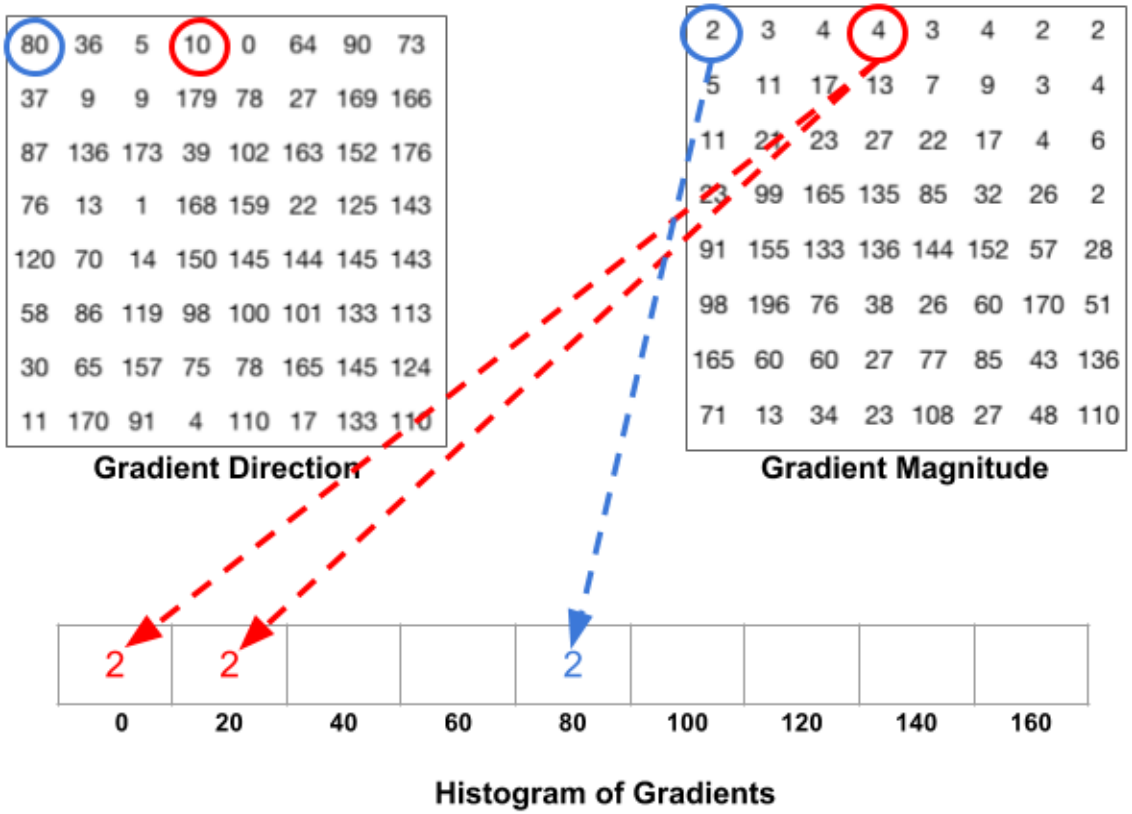
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

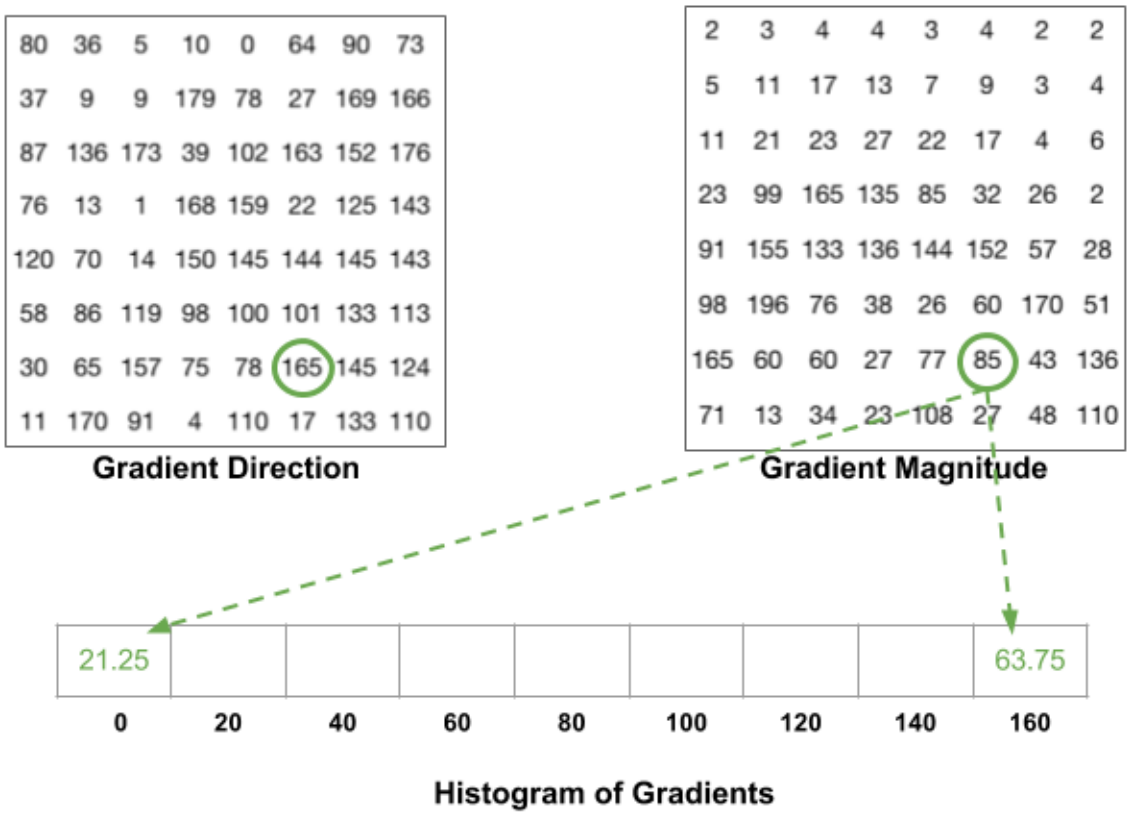
Gradient Direction

Center : The RGB patch and gradients represented using arrows.
Right : The gradients in the same patch represented as numbers

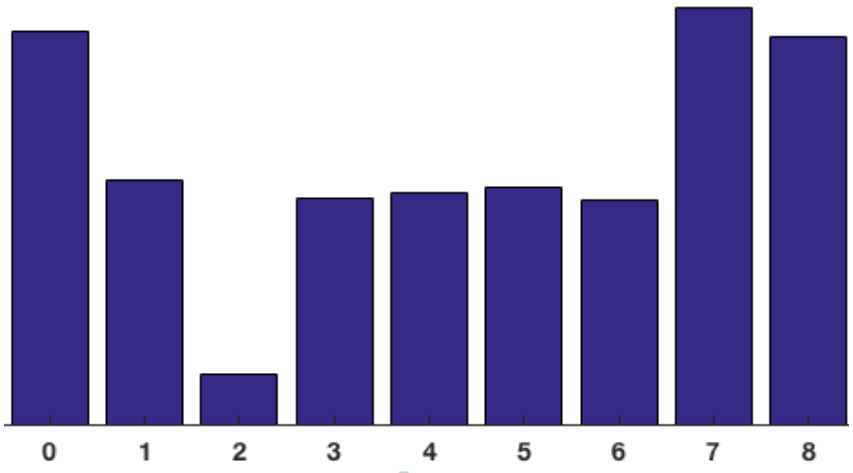
HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



HOG : Step 3 : Calculate Histogram of Gradients in 8x8 cells



HOG : Step 4 : 16x16 Block Normalization

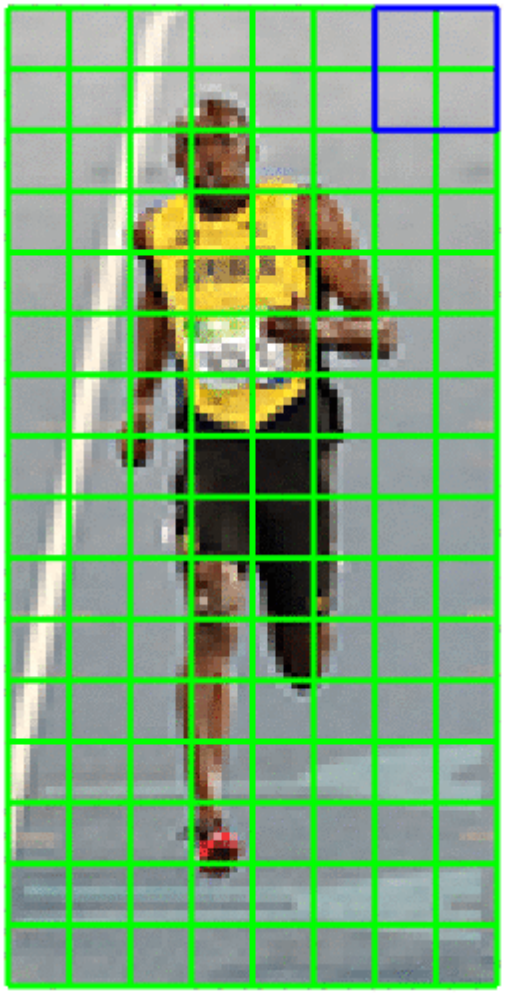
HOG : Step 4 : 16x16 Block Normalization

HOG : Step 4 : 16x16 Block Normalization

HOG : Step 4 : 16x16 Block Normalization

HOG : Step 4 : 16x16 Block Normalization

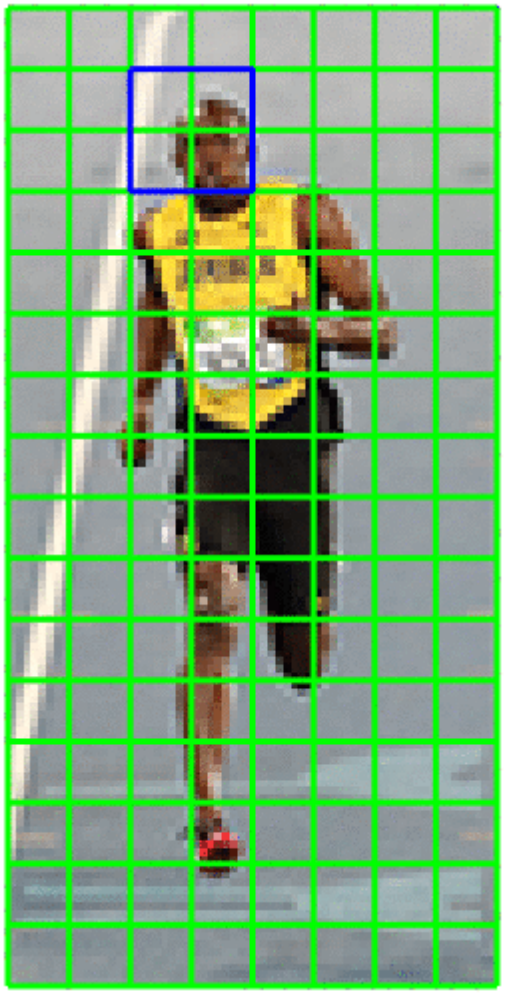
HOG : Step 4 : 16x16 Block Normalization



HOG : Step 4 : 16x16 Block Normalization

HOG : Step 4 : 16x16 Block Normalization

HOG : Step 4 : 16x16 Block Normalization

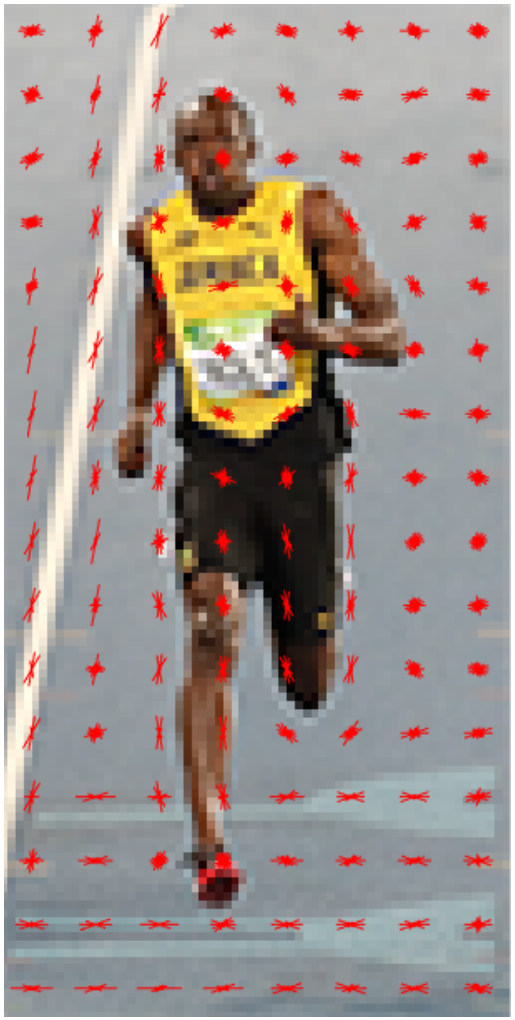


HOG : Step 5 : Calculate the HOG feature vector

To calculate the final feature vector for the entire image patch, the 36x1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate

1. How many positions of the 16x16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
2. Each 16x16 block is represented by a 36x1 vector. So when we concatenate them all into one gaint vector we obtain a 36x105 = 3780 dimensional vector.

HOG : Visualizing Histogram of Oriented Gradients



Frame differencing

```
import cv2

# Compute the frame differences
def frame_diff(prev_frame, cur_frame, next_frame):
    # Difference between the current frame and the next frame
    diff_frames_1 = cv2.absdiff(next_frame, cur_frame)

    # Difference between the current frame and the previous frame
    diff_frames_2 = cv2.absdiff(cur_frame, prev_frame)

    return cv2.bitwise_and(diff_frames_1, diff_frames_2)
```

Frame differencing

```
# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    return gray
```

Frame differencing

```
if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the images
    scaling_factor = 0.5

    # Grab the current frame
    prev_frame = get_frame(cap, scaling_factor)

    # Grab the next frame
    cur_frame = get_frame(cap, scaling_factor)

    # Grab the frame after that
    next_frame = get_frame(cap, scaling_factor)
```

Frame differencing

```
if __name__=='__main__':
 ....

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Display the frame difference
        cv2.imshow('Object Movement', frame_diff(prev_frame,
                cur_frame, next_frame))

        # Update the variables
        prev_frame = cur_frame
        cur_frame = next_frame

        # Grab the next frame
        next_frame = get_frame(cap, scaling_factor)

        # Check if the user hit the 'Esc' key
        key = cv2.waitKey(10)
        if key == 27:
            break

    # Close all the windows
cv2.destroyAllWindows()
```

Background substraction

```
import cv2
import numpy as np

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame
```

Background substraction

```
if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the background subtractor object
    bg_subtractor = cv2.createBackgroundSubtractorMOG2()

    # Define the number of previous frames to use to learn.
    # This factor controls the learning rate of the algorithm.
    # The learning rate refers to the rate at which your model
    # will learn about the background. Higher value for
    # "history" indicates a slower learning rate. You can
    # play with this parameter to see how it affects the output.
    history = 100

    # Define the learning rate
    learning_rate = 1.0/history
```

Background substraction

```
if __name__=='__main__':
    .....

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Grab the current frame
        frame = get_frame(cap, 0.5)

        # Compute the mask
        mask = bg_subtractor.apply(frame, learningRate=learning_rate)

        # Convert grayscale image to RGB color image
        mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

        # Display the images
        cv2.imshow('Input', frame)
        cv2.imshow('Output', mask & frame)

        # Check if the user hit the 'Esc' key
        c = cv2.waitKey(10)
        if c == 27:
            break

    # Release the video capture object
    cap.release()

    # Close all the windows
cv2.destroyAllWindows()
```

CAMShift

- Color space tracker : define the color first.
- CAMShift : consider a region of interest
- Select the region

CAMShift

```
import cv2
import numpy as np

# Define a class to handle object tracking related functionality
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Initialize the video capture object
        self.cap = cv2.VideoCapture(0)
        # Capture the frame from the webcam
        _, self.frame = self.cap.read()
        # Scaling factor for the captured frame
        self.scaling_factor = scaling_factor
        # Resize the frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)
        # Create a window to display the frame
        cv2.namedWindow('Object Tracker')
        # Set the mouse callback function to track the mouse
        cv2.setMouseCallback('Object Tracker', self.mouse_event)
        # Initialize variable related to rectangular region selection
        self.selection = None
        # Initialize variable related to starting position
        self.drag_start = None
        # Initialize variable related to the state of tracking
        self.tracking_state = 0
```

CAMShift

```
# Define a method to track the mouse events
def mouse_event(self, event, x, y, flags, param):
    # Convert x and y coordinates into 16-bit numpy integers
    x, y = np.int16([x, y])

    # Check if a mouse button down event has occurred
    if event == cv2.EVENT_LBUTTONDOWN:
        self.drag_start = (x, y)
        self.tracking_state = 0

    # Check if the user has started selecting the region
    if self.drag_start:
        if flags & cv2.EVENT_FLAG_LBUTTON:
            # Extract the dimensions of the frame
            h, w = self.frame.shape[:2]
            # Get the initial position
            xi, yi = self.drag_start
            # Get the max and min values
            x0, y0 = np.maximum(0, np.minimum([xi, yi], [x, y]))
            x1, y1 = np.minimum([w, h], np.maximum([xi, yi], [x, y]))
            # Reset the selection variable
            self.selection = None
            # Finalize the rectangular selection
            if x1-x0 > 0 and y1-y0 > 0:
                self.selection = (x0, y0, x1, y1)
```

CAMShift

```
        else:
            # If the selection is done, start tracking
            self.drag_start = None
            if self.selection is not None:
                self.tracking_state = 1

# Method to start tracking the object
def start_tracking(self):
    # Iterate until the user presses the Esc key
    while True:
        # Capture the frame from webcam
        _, self.frame = self.cap.read()

        # Resize the input frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)

        # Create a copy of the frame
        vis = self.frame.copy()

        # Convert the frame to HSV colorspace
        hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)

        # Create the mask based on predefined thresholds
        mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                    np.array((180., 255., 255.)))
```

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
    .....

        # Check if the user has selected the region
        if self.selection:
            # Extract the coordinates of the selected rectangle
            x0, y0, x1, y1 = self.selection

            # Extract the tracking window
            self.track_window = (x0, y0, x1-x0, y1-y0)

            # Extract the regions of interest
            hsv_roi = hsv[y0:y1, x0:x1]
            mask_roi = mask[y0:y1, x0:x1]

            # Compute the histogram of the region of
            # interest in the HSV image using the mask
            hist = cv2.calcHist( [hsv_roi], [0], mask_roi,
                    [16], [0, 180] )
```

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
    .....


            # Normalize and reshape the histogram
            cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX);
            self.hist = hist.reshape(-1)

            # Extract the region of interest from the frame
            vis_roi = vis[y0:y1, x0:x1]

            # Compute the image negative (for display only)
            cv2.bitwise_not(vis_roi, vis_roi)
            vis[mask == 0] = 0
```

CAMShift

```
# Method to start tracking the object
def start_tracking(self):
  ....
        # Check if the system in the "tracking" mode
        if self.tracking_state == 1:
            # Reset the selection variable
            self.selection = None
            # Compute the histogram back projection
            hsv_backproj = cv2.calcBackProject([hsv], [0],
                    self.hist, [0, 180], 1)
            # Compute bitwise AND between histogram
            # backprojection and the mask
            hsv_backproj &= mask
            # Define termination criteria for the tracker
            term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
                    10, 1)
```

CAMShift

```
    # Method to start tracking the object
    def start_tracking(self):
      ....
                # Apply CAMShift on 'hsv_backproj'
                track_box, self.track_window = cv2.CamShift(hsv_backproj,
                        self.track_window, term_crit)
                # Draw an ellipse around the object
                cv2.ellipse(vis, track_box, (0, 255, 0), 2)

            # Show the output live video
            cv2.imshow('Object Tracker', vis)
            # Stop if the user hits the 'Esc' key
            c = cv2.waitKey(5)
            if c == 27:
                break

        # Close all the windows
        cv2.destroyAllWindows()

if __name__ == '__main__':
    ObjectTracker().start_tracking()
```

Tests & Features

IML

Cédric Buche

ENIB

29 août 2019