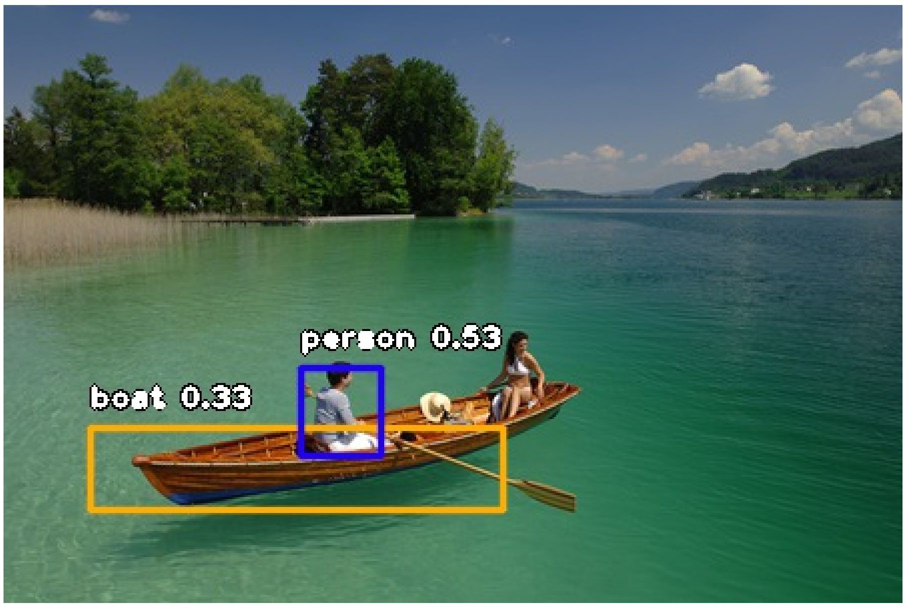
//

Machine Learning VIII

maXbox Starter 75 – Object Detection

From Document to Recognition?
Detect the Rect!

This tutor puts a trip to the kingdom of object recognition with computer vision knowledge and an image classifier.
Object detection has been witnessing a rapid revolutionary change in some fields of computer vision. Its involvement in the combination of object classification as well as object recognition makes it one of the most challenging topics in the domain of machine learning & vision.



First we need a library with modules. **ImageAI** is a Python library built to empower developers to build applications and systems with self-contained deep learning and Computer Vision capabilities using a few lines of straight forward code. But to use ImageAI you need to install a few dependencies namely:

- TensorFlow
- OpenCV
- Keras and ImageAI itself to install with $ pip3 install imageAI

Now download the TinyYOLOv3 model file (33.9 MB) that contains a pretrained classification model that will be used for object detection:

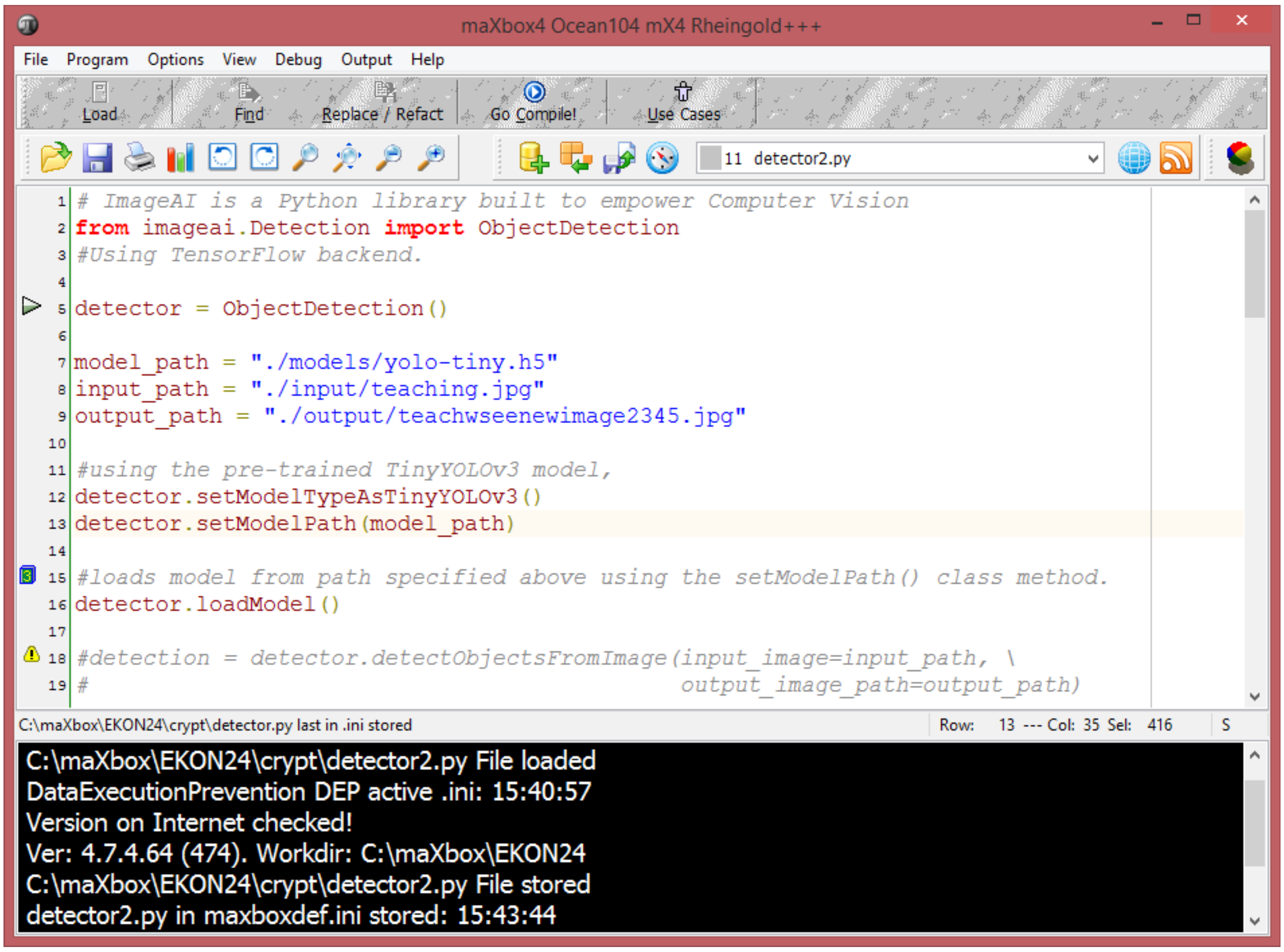
https://sourceforge.net/projects/maxbox/files/Examples/EKON/EKON24/ImageDetector/yolo-tiny.h5/download

Then we need 3 necessary folders.

- input\
- models\yolo-tiny.h5
- output

Now put an image for detection in the input folder, for example: teaching.jpg

Open now your preferred text editor for writing Python code (in my case maXbox) and create a new file *detector.py* or some valid file name.


maXbox4 Ocean104 mX4 Rheingold+++
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compile!
Use Cases
11 detector2.py

```
 1 # ImageAI is a Python library built to empower Computer Vision
 2 from imageai.Detection import ObjectDetection
 3 #Using TensorFlow backend.
 4
 5 detector = ObjectDetection()
 6
 7 model_path = "./models/yolo-tiny.h5"
 8 input_path = "./input/teaching.jpg"
 9 output_path = "./output/teachwseenewimage2345.jpg"
10
11 #using the pre-trained TinyYOLOv3 model,
12 detector.setModelTypeAsTinyYOLOv3()
13 detector.setModelPath(model_path)
14
15 #loads model from path specified above using the setModelPath() class method.
16 detector.loadModel()
17
18 #detection = detector.detectObjectsFromImage(input_image=input_path, \
19 #                                           output_image_path=output_path)
```

```
C:\maXbox\EKON24\crypt\detector.py last in .ini stored    Row: 13 --- Col: 35 Sel: 416   S
```

```
C:\maXbox\EKON24\crypt\detector2.py File loaded
DataExecutionPrevention DEP active .ini: 15:40:57
Version on Internet checked!
Ver: 4.7.4.64 (474). Workdir: C:\maXbox\EKON24
C:\maXbox\EKON24\crypt\detector2.py File stored
detector2.py in maxboxdef.ini stored: 15:43:44
```

In line 2 we import *ObjectDetection* class from the **ImageAI** library.

```
from imageai.Detection import ObjectDetection
```

As the next thing we create an instance of the class *ObjectDetection*, as shown in line 5 above:

```
detector = ObjectDetection()
```

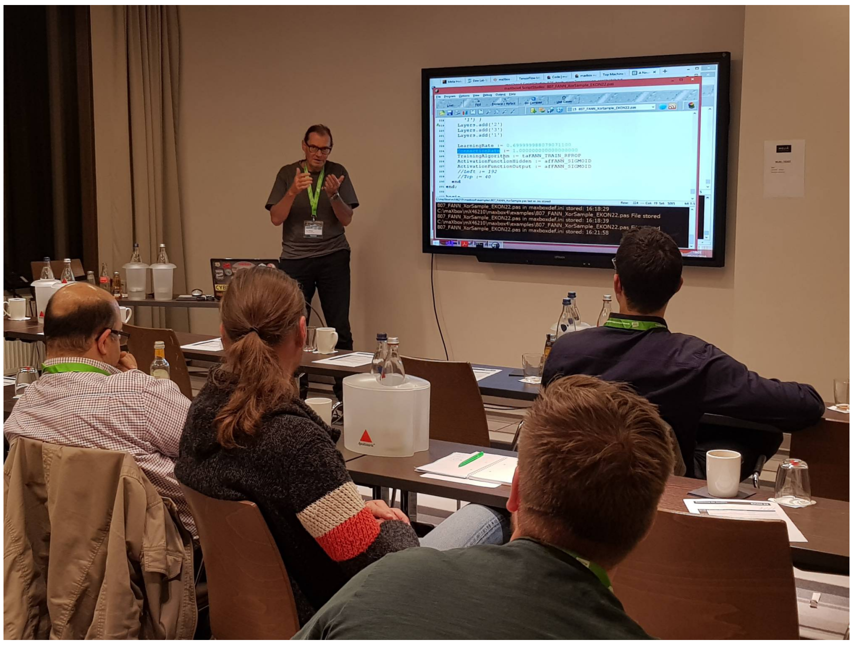
It goes on with the declaration of the previous created paths:

```
model_path = "./models/yolo-tiny.h5"
input_path = "./input/teaching.jpg"
output_path = "./output/the_newimage.jpg"
```

In this tutorial, as I mentioned we'll be using the pre-trained TinyYOLOv3 model, so I use the *setModelTypeAsTinyYOLOv3*() function to load our model:

```
#using the pre-trained TinyYOLOv3 model,
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath(model_path)
```

```
#loads model from path specified above using the setModelPath() class method.
detector.loadModel()
```



To detect only some of the objects above, I will need to call the *CustomObjects* method and set the name of the object(s) we want to detect to through. The rest are False by default. In our example, we detect customized only person, laptop and bottle. The boat is to test some negative test (maybe it find some message in the bottle with a boat in it ;-)).

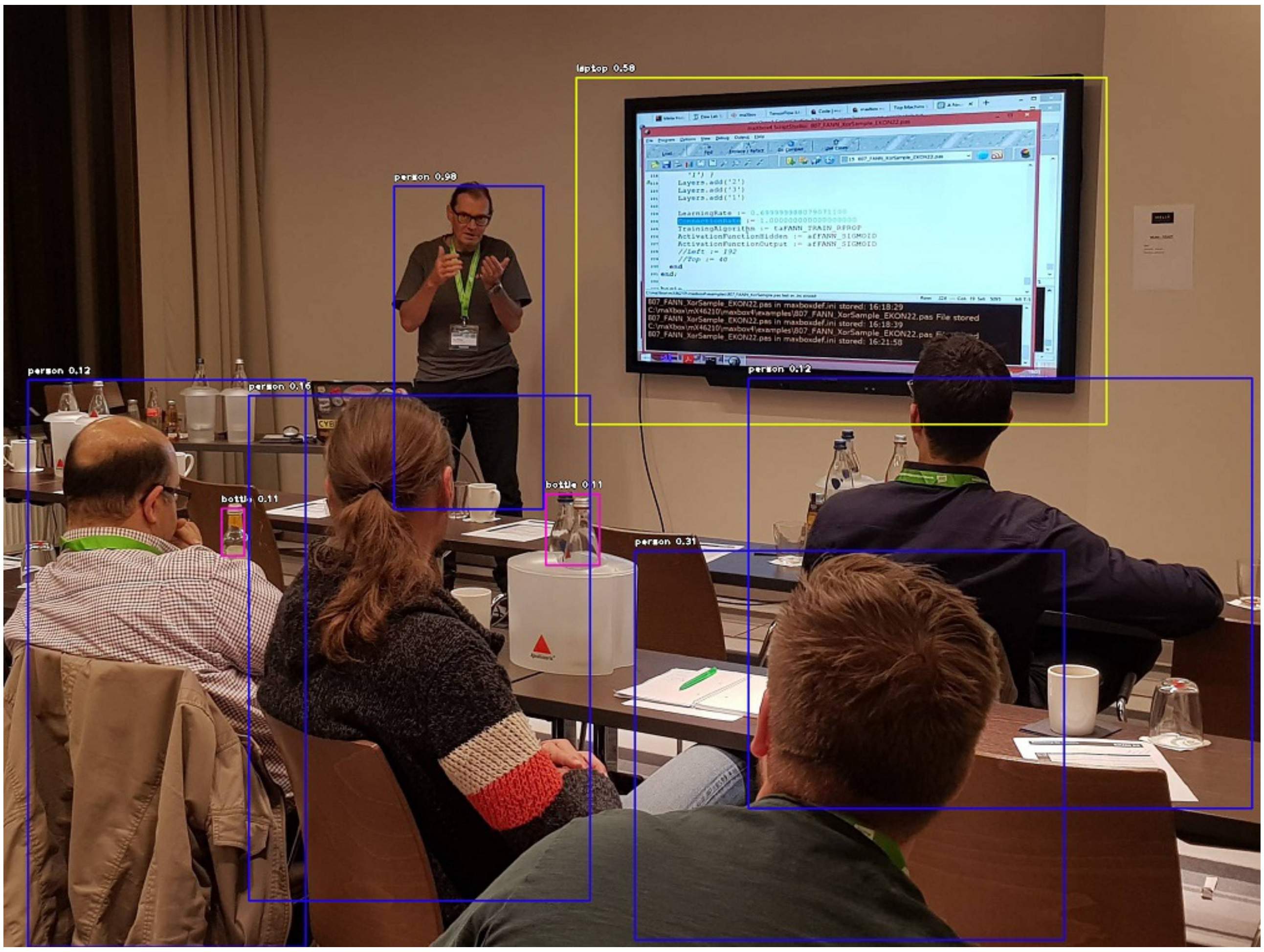
```
custom= detector.CustomObjects(person=True,boat=True,laptop=True,bottle=True)
detections = detector.detectCustomObjectsFromImage(custom_objects=custom, \
                  input_image=input_path, output_image_path=output_path,\
                                          minimum_percentage_probability=10)
```

Another reason for the custom instance is that I can define the threshold to find things. Unlike the normal *detectObjectsFromImage*() function, this needs an extra parameter which is "custom_object" which accepts the dictionary returned by the *CustomObjects*() function. In the sample below, we set the detection function to report only detections we want:

```
for eachItem in detections:
    print(eachItem["name"] , " : ", eachItem["percentage_probability"])
```

```
laptop  :  57.53162503242493
bottle  :  10.687477886676788
bottle  :  11.373373866081238
person  :  11.838557571172714
person  :  12.098842114210129
person  :  15.951324999332428
person  :  31.1357319355011
person  :  98.0242371559143
image detector compute ends...
```

With the parameter minimum_percentage_probability=30 it could not find the 2 bottles, and yes we got an output!:



So we get 8 objects with color frames and corresponding probability. The 2 bottles by lila color and astonishing the yellow frame is the laptop. Funny but really effective!

This is the function (detectCustomObjectsFromImage) that performs object detection task after the model as loaded. It can be called many times to detect objects in any number of images.

Script detector2.py

```
# ImageAI is a Python library built to empower Computer Vision
from imageai.Detection import ObjectDetection
#Using TensorFlow backend.

detector = ObjectDetection()

model_path = "./models/yolo-tiny.h5"
input_path = "./input/teaching.jpg"
output_path = "./output/teachwseenewimage2345.jpg"

#using the pre-trained TinyYOLOv3 model,
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath(model_path)

detector.loadModel()
```

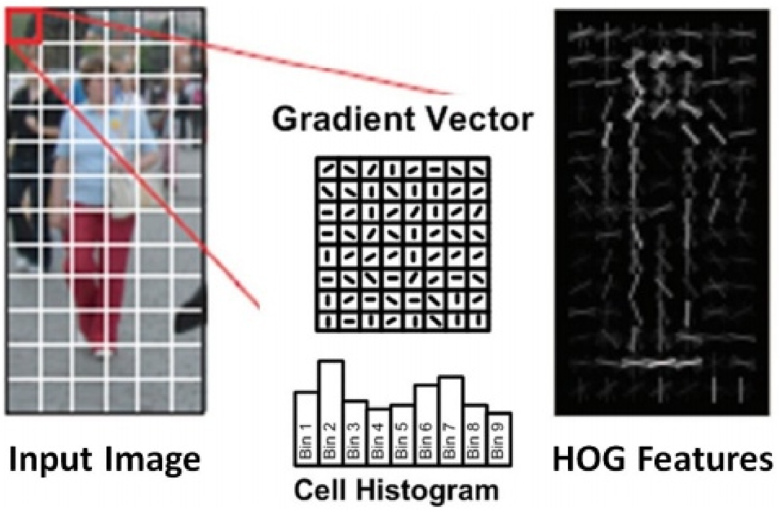
```
#detection = detector.detectObjectsFromImage(input_image=input_path, \
#                                            output_image_path=output_path)

custom= detector.CustomObjects(person=True,boat=True, laptop=True,bottle=True)
detections = detector.detectCustomObjectsFromImage(custom_objects=custom, \
                  input_image=input_path, output_image_path=output_path,\
                                          minimum_percentage_probability=10)

for eachItem in detections:
    print(eachItem["name"] , " : ", eachItem["percentage_probability"])

print('image detector compute ends...')

#https://stackabuse.com/object-detection-with-imageai-in-python/
#https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo-tiny.h5
#https://imageai.readthedocs.io/en/latest/detection/index.html
```

There are 80 possible objects that you can detect with the ObjectDetection class, and they are as seen below (not ordered).

```
    person,   bicycle,   car,   motorcycle,   airplane,
    bus,   train,   truck,   boat,   traffic light,   fire hydrant,   stop_sign,
    parking meter,   bench,   bird,   cat,   dog,   horse,   sheep,   cow,
elephant,   bear,   zebra,   giraffe,   backpack,   umbrella,   handbag,   tie,
suitcase,   frisbee,   skis,   snowboard,   sports ball,   kite,   baseball bat,
baseball glove,   skateboard,   surfboard,   tennis racket,   bottle,   wine
glass,   cup,   fork,   knife,   spoon,   bowl,   banana,   apple,   sandwich,
orange,   broccoli,   carrot,   hot dog,   pizza,   donot,   cake,   chair,
couch,   potted plant,   bed,   dining table,   toilet,   tv,   laptop,
mouse,   remote,   keyboard, cell phone,   microwave, oven,   toaster, sink,
refrigerator, book, clock, vase, scissors, teddy bear, hair dryer,  toothbrush.
```

To detect only some of the objects above, you will need to call the *CustomObjects* function and set the name of the object(s) you want to detect.

Note:
Histogram of oriented gradients (HOG) is basically a feature descriptor that is used to detect objects in image processing and other computer vision techniques. The Histogram of oriented gradients descriptor technique includes occurrences of gradient orientation in localised portions of an image, such as detection window, region of interest (ROI), among others. Advantage of HOG-like features is their simplicity, and it is easier to understand information they carry.



Appendix: See also two other classifiers

SGDClassifier
LogisticRegressionCV

```
SGDClassifier
    incrementally trained logistic regression (when given parameter loss="log").
LogisticRegressionCV
    Logistic regression with built-in cross validation
```

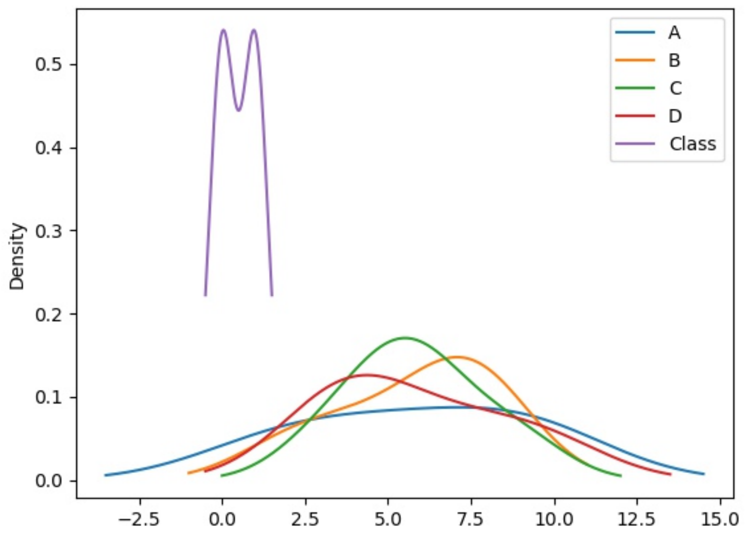
Notes:

The underlying C implementation uses a random number generator to select features when fitting the model. It is thus not uncommon, to have slightly different results for the same input data. If that happens, try with a smaller tol parameter or set random state to 0.

Mathematically, a histogram is a mapping of bins (intervals or numbers) to frequencies. More technically, it can be used to approximate a probability density function (PDF) of the underlying variable that we see later on. Moving on from a frequency table above (density=**False** counts at y-axis), a true histogram first <bins> the range of values and then counts the number of values that fall into each bin or interval. A plot of a histogram uses its bin edges on the x-axis and the corresponding frequencies on the y-axis.

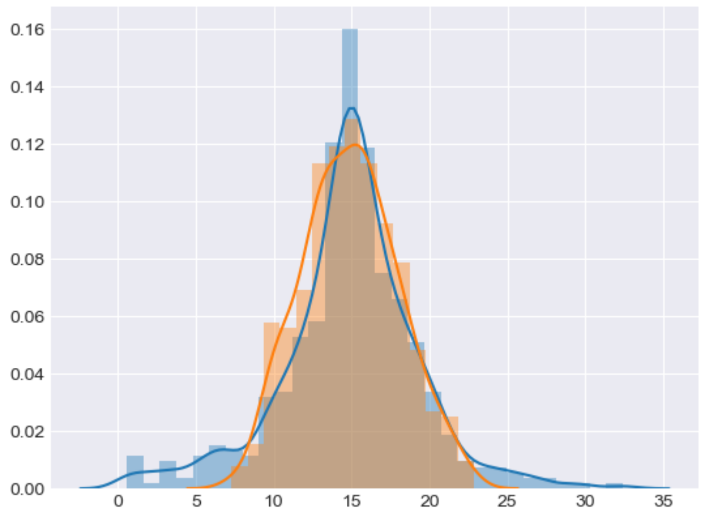
Sticking with the Pandas library, you can create and overlay density plots using plot.kde(), which is available for both [Series] and [DataFrame] objects.

```
df.iloc[0:,0:4].plot.kde()
```



This is also possible for our binary targets to see a probabilistic distribution of the target class values (labels in supervised learning): **[0. 0. 1. 1. 0. 1.]**
Consider at last a sample of floats drawn from the Laplace and Normal distribution together. This distribution graph has fatter tails than a normal distribution and has two descriptive parameters (location and scale):

```
>>> d = np.random.laplace(loc=15, scale=3, size=500)
>>> d = np.random.normal(loc=15, scale=3, size=500)
```



The script and data can be found:

http://www.softwareschule.ch/examples/detector2.htm
https://sourceforge.net/projects/maxbox/files/Examples/EKON/EKON24/ImageDetector/

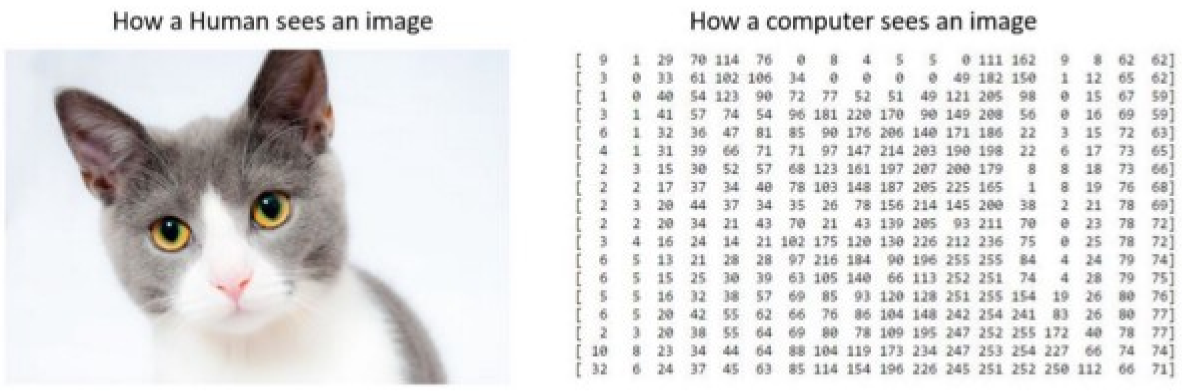
http://www.softwareschule.ch/examples/classifier_compare2confusion2.py.txt

Author: Max Kleiner

Ref:
    http://www.softwareschule.ch/box.htm
    https://scikit-learn.org/stable/modules/
    https://realpython.com/python-histograms/

    https://imageai.readthedocs.io/en/latest/detection/index.html

Doc:
    https://maxbox4.wordpress.com



```
[   9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
[   3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
[   1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
[   3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
[   6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
[   4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
[   2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
[   2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
[   2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
[   2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
[   3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
[   6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
[   6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
[   5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
[   6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
[   2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
[  10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
[  32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]
```