Topic 5 - Image Pre-processing and Feature Extraction

Aim of the Session

- Learn about the most basic methods that help us clean images
- Understand how images are converted into a set of features used for machine learning purposes

Resources for the Lecture

Websites

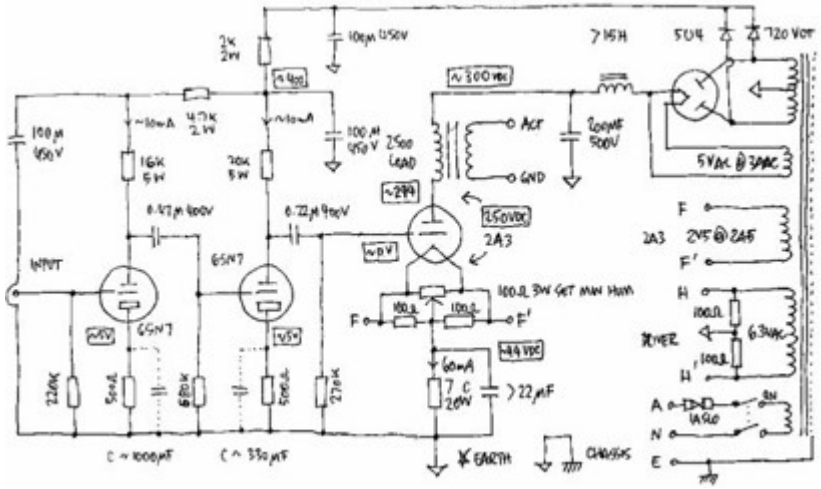
- https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html (https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html)
- https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html (https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html)
- https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients (https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients)
- https://www.learnopencv.com/histogram-of-oriented-gradients/ (https://www.learnopencv.com/histogram-of-oriented-gradients/)
- https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework (https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework)
- https://realpython.com/traditional-face-detection-python/ (https://realpython.com/traditional-face-detection-python/)
- https://github.com/opencv/opencv/tree/master/data/haarcascades (https://github.com/opencv/opencv/tree/master/data/haarcascades)

Papers

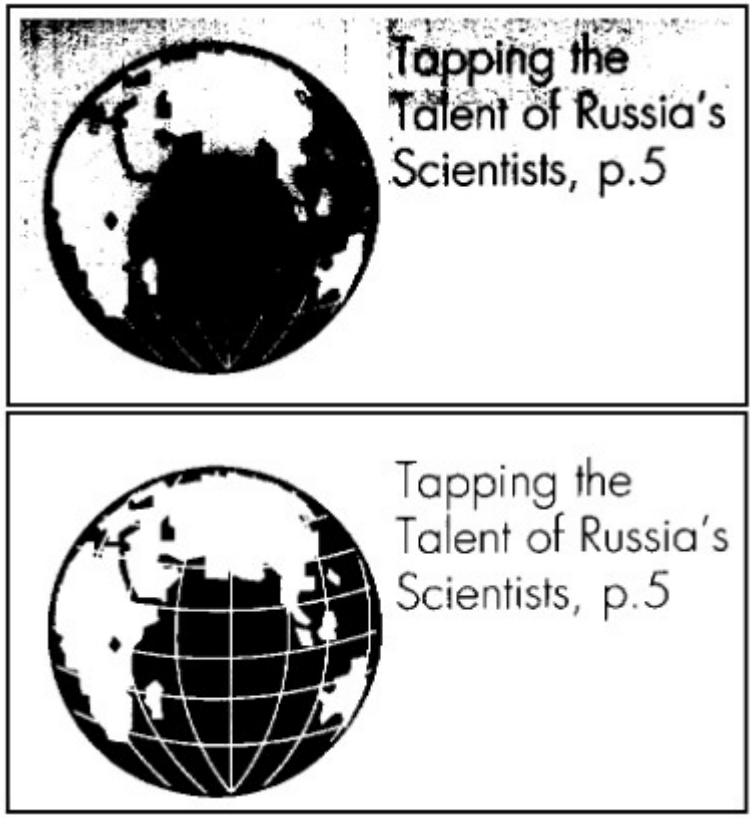
- Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys. Man. Cyber. 9 (1): 62–66.
- N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” Computer Vision and Pattern Recognition (CVPR), vol. I, pp. 886–893, 2005.
- D. G. Lowe, “Object recognition from local scale-invariant features,” International Conference on Computer Vision (ICCV), vol. 2, no. 8, pp. 1150–1157, 1999.
- H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” Lecture Notes in Artificial Intelligence, vol. 3951, pp. 404–417, 2006.

Image Preprocessing

- Not all images are perfect, especially in the document image analysis


2K 2W
10µF 450V
>15H
5U4
720 VCT
~300 VAC
100µ 450V
~10mA
4.7K 2W
16K 5W
20K 5W
100µ 450V
2500 LEAD
ACT
GND
2000µF 500V
5VAC @ 3AMP
~299
250VDC
2A3
0.47µ 600V
0.22µ 400V
~0V
INPUT
6SN7
F
F'
100Ω 3W SET MAN HUM
100Ω
100Ω
60mA
7 C 20W
>22µF
+44 VDC
2A3 2V5 @ 2A5
H
H'
100Ω
POWER
6.3VAC
A
N
E
1A90
3W
220K
500Ω
680K
500Ω
270K
~45V
~45V
C ~ 1000µF
C ~ 330µF
EARTH
CHASSIS

THRESHOLDING/BINARISATION



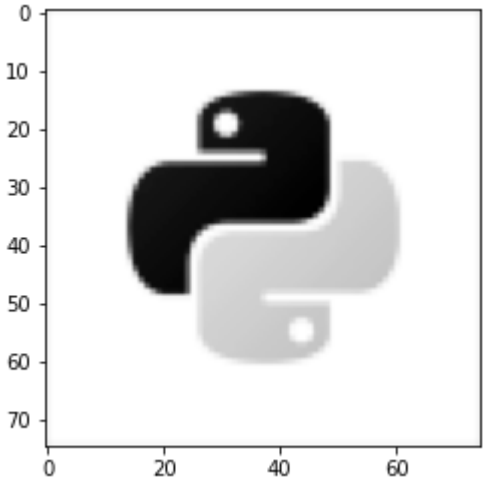
- Useful to improve the quality of an image and to refine shapes

How to binarise an image?

First, load the image in grayscale mode

In [2]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
img = cv2.imread("data/logo.png",0)
plt.imshow(img, cmap='Greys_r')
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x1b06ac768d0>
```

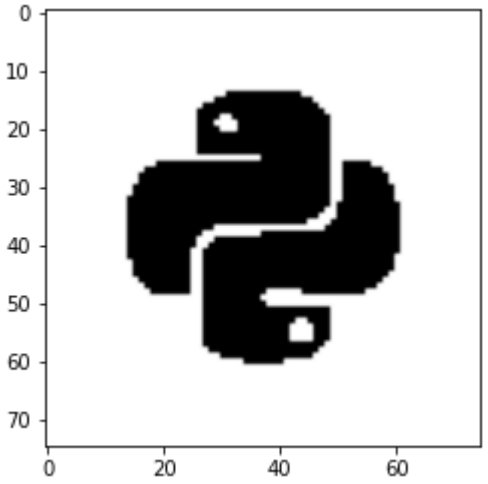


Then, you need a function that traverses over **all** pixels and compares their values against a threshold. If the value found is larger than the threshold, then the values is changed to 255, otherwise, it is changes to 0.

In [3]:
```
## We define a function called binarise with two params, the image and the threshold
def binarise(img, threshold):
    '''This function biniarises an image in a numpy array according to the specified thr
eshold'''
    img_bin = img.copy()
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            if img[x,y]<threshold:
                img_bin[x,y]=0
            else:
                img_bin[x,y]=255
    plt.imshow(img_bin, cmap='Greys_r')
    return img_bin
```

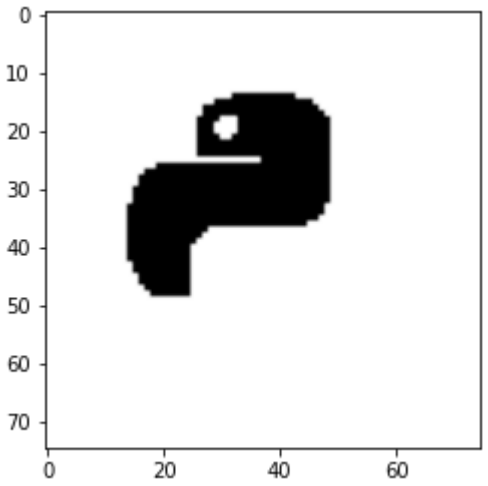
Now, we can call our function using the loaded image

In [4]:
```
## Use this cell to execute the binarise function and show both pythons in black.
img_bin1 = binarise(img, 220)
```



If we change the threshold value, then funny things can happen!

In [5]:
```
## Use this cell to execute the binarise function to show only the dark python and "disa
ppear" the light one.
img_bin2 = binarise(img, 200)
```



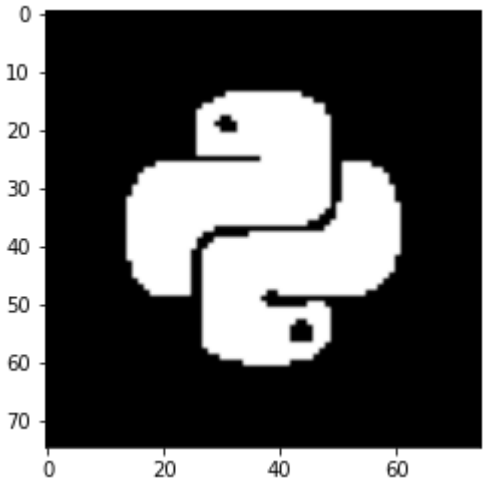
The cv2 option

Click here (https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html) to see a very comprehensive tutorial on how to use the binarisation function contained in `OpenCV` .

Then apply the function to do the same things that you did with your function.

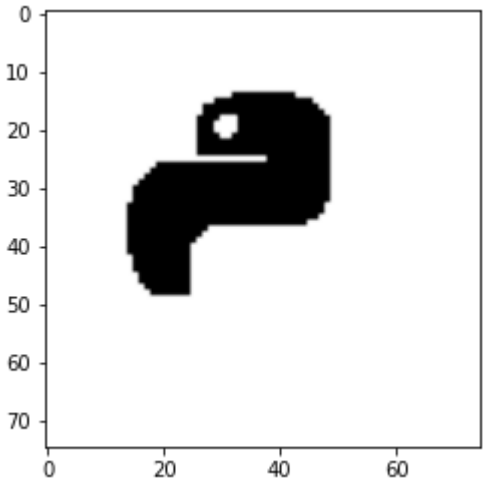
In [6]:
```
## Use this cell to execute the cv2.threshold function and show both pythons in black.
_,thresh1 = cv2.threshold(img,220,255,cv2.THRESH_BINARY_INV)
plt.imshow(thresh1, cmap='Greys_r')
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x1b06ce3cef0>
```



In [7]:
```
## Use this cell to execute the cv2.threshold function and show only the dark and "disap
pear" the light one.
_,thresh2 = cv2.threshold(img,190,255,cv2.THRESH_BINARY)
plt.imshow(thresh2, cmap='Greys_r')
```

Out[7]: <matplotlib.image.AxesImage at 0x1b06cea8438>



Saving the image

You can save any of the images produced in this tutorial by using either the `cv2.imwrite` or the `plt.imsave` command.

For instance, to save the binarised image, use the following command:

In [8]:
```
cv2.imwrite('thresh2.png',thresh2)
```

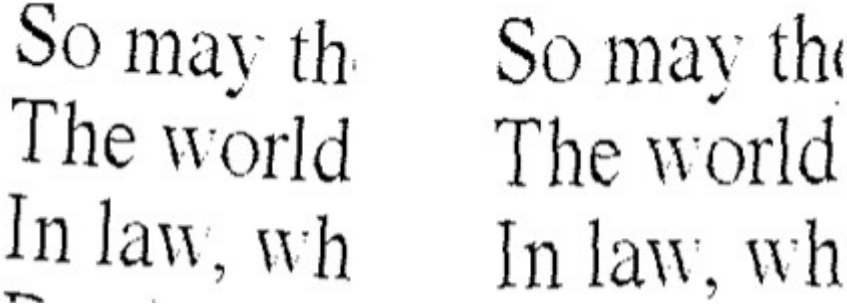
Out[8]:
```
True
```

I recommend to use `cv2.imwrite` , given that the `plt` option will save it with the yellow background by default.

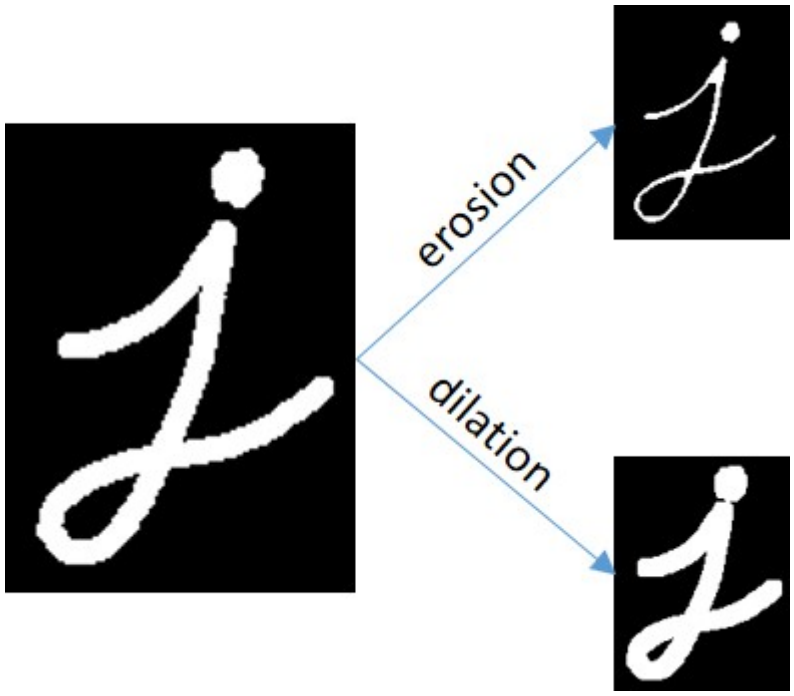
Bonus: Can you "disappear" only the top Python?

Other pre-processing techniques

Skew Correction



Erosion and Dilation



Opening

Closing

Feature Extraction

Features are the most important component of data science

Think about any data science problem that you have faced so far

You have to analyse a series of data records/entries, each possessing a fixed amount of attributes to describe them

RESULTS TABLE

Identification | 0 RECORDS SELECTED | VIEW OPTIONS | ACTIONS

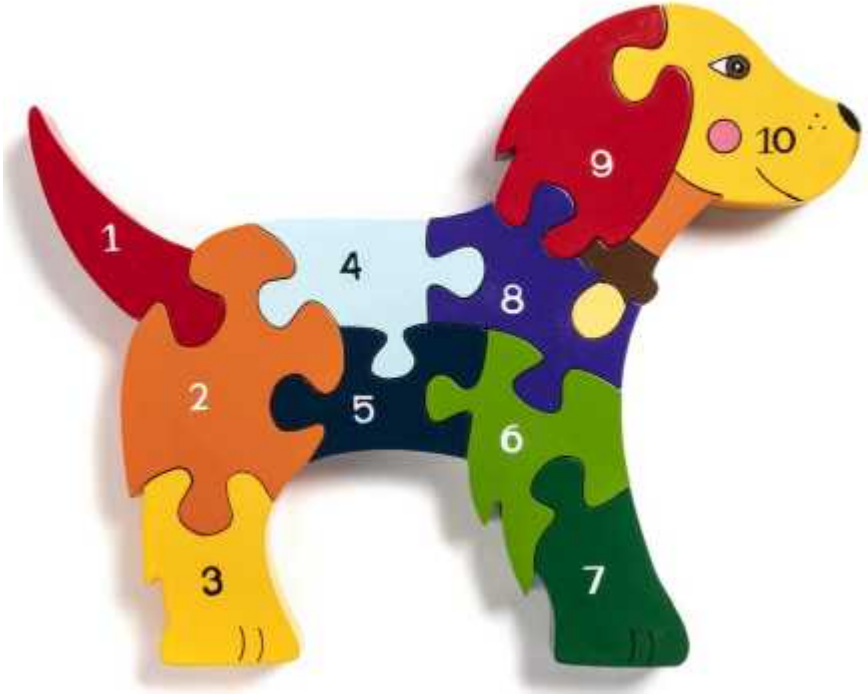
Wine ID	Name	Description	Wine Type	Vintage	Winery	
34707	Alentejo Conventual	Some plum and cherry fla...	Red	1992	Adega Cooperativa de Po...	Portu
34708	Alentejo Monte Velho	This shows supple, flesh...	Red	1992	Herdade do Esporao	Portu
34709	Alentejo Tinto da Anforna	Smells like a melange of s...	Alentejo, Red	1992	J.P. Vinhos	Portu
34710	Alentejo Tinto da Talha	Soft, generous and fruity,...	Alentejo, Red	1992	Roquevale	Portu
34711	Alentejo Vinha do Monte	Here's a bruiser. Deeply ...	Red	1992	Sogrape	Portu
34712	Alentejo	Fruity, concentrated and ...	Red	1992	Adega Cooperativa de B...	Portu
34713	Alliance California	Rustic, rough-and-tumble...	Red, Rhone Blend	1992	R.H. Phillips	Other
34714	Aloxe-Corton	Simple and appealing, wit...	Burgundy Cote de Be...	1992	Bouchard Pere & Fils	Burgu
34715	Aloxe-Corton	Interesting floral and rhub...	Burgundy Cote de Be...	1992	Jaffelin	Burgu
34716	Aloxe-Corton	Light and sweet-tasting, ...	Burgundy Cote de Be...	1992	Louis Latour	Burgu

PAGE 1 OF 2854

1-20 OF 57076

Features in images

How do you assemble a jigsaw puzzle?



Can we project the same logic to the computer?

Can we project the same logic to data science as a whole?

Pattern Recognition

The computer will look for specific **patterns** which are

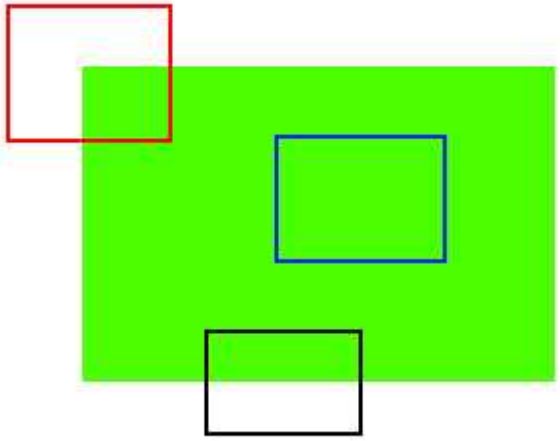
- unique
- easy to track
- easy to compare

Which are the **best** features to do so!

Take a look at this image and find the features:

- A and B are flat surfaces and difficult to find as an exact match
- C and D are simpler (edges of the building), however their exact location is difficult to find
- E and F are corners of the building, and can be easily found based on their **patch**

Taking a simpler example you can see that **corners** are usually the most intuitive **structural features**



There are other good features, such as blobs, changes of intensity, etc.

Feature detection and extraction algorithms

Last week, we saw that pixels can be used to create datasets

Nonetheless, sometimes it is better to **extract** features from images, which results in a 2D feature vector that can be used instead!

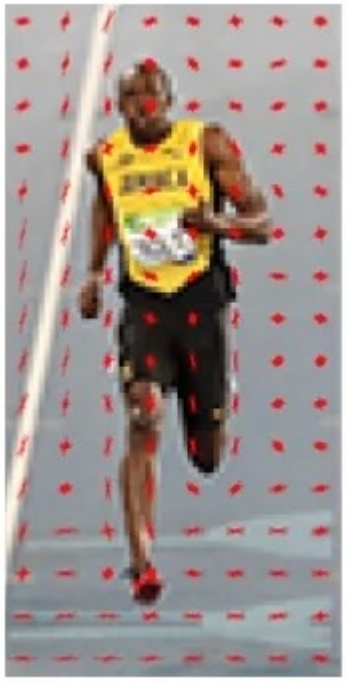
Today, we will see two example, one for general images and another for faces, which also result in detection

Histogram of Oriented Gradients (HOG)

The technique counts occurrences of gradient orientation in localised portions of an image

In simple terms: It converts a $64 \times 128 \times 3$ image into a **feature vector** of length 3780

- Why this size?



- The method calculates the x and y gradients of the image, as well as the magnitude
- Then the image is divided in patches and the gradients are calculated

In [9]:

```
## Don't print warnings
import warnings;
warnings.simplefilter('ignore')
## Obtaining the HOG gradients of an image
from skimage import feature
class HOG:
    def __init__(self, orientations = 9, pixelsPerCell = (8, 8),
        cellsPerBlock = (2, 2), transform = False):
        self.orienations = orientations
        self.pixelsPerCell = pixelsPerCell
        self.cellsPerBlock = cellsPerBlock
        self.transform = transform
    def describe(self, image):
        hist = feature.hog(image, orientations = self.orienations,
            pixels_per_cell = self.pixelsPerCell,
            cells_per_block = self.cellsPerBlock,
            transform_sqrt = self.transform)
        return hist
img_usain = cv2.imread('data/usain.jpg',0)
hog = HOG(orientations = 9, pixelsPerCell = (8, 8), cellsPerBlock = (2, 2), transform =
True)
hist = hog.describe(img_usain)
print('HOG Features for the image:')
print(hist)
print('Size of the HOG Features for the image:', hist.shape)
```

```
HOG Features for the image:
[0.13199286 0.03729767 0.1958527  ... 0.         0.         0.        ]
Size of the HOG Features for the image: (3780,)
```

Notice that in this case, we obtain a fixed-size feature vector provided that the input has the same size

Approaches like this are much better suited for machine learning compared to classical feature extractors such as Harris Corners). **WHY?**

Still, classical feature extractors are widely used since these are capable to represent **structure**

HOW MANY MORE FEATURE EXTRACTORS EXIST??

MANY, MANY, MAAAAAAAAANY OTHERS!!!

Features for Face Recognition (+ detection)

The human face innately has features!

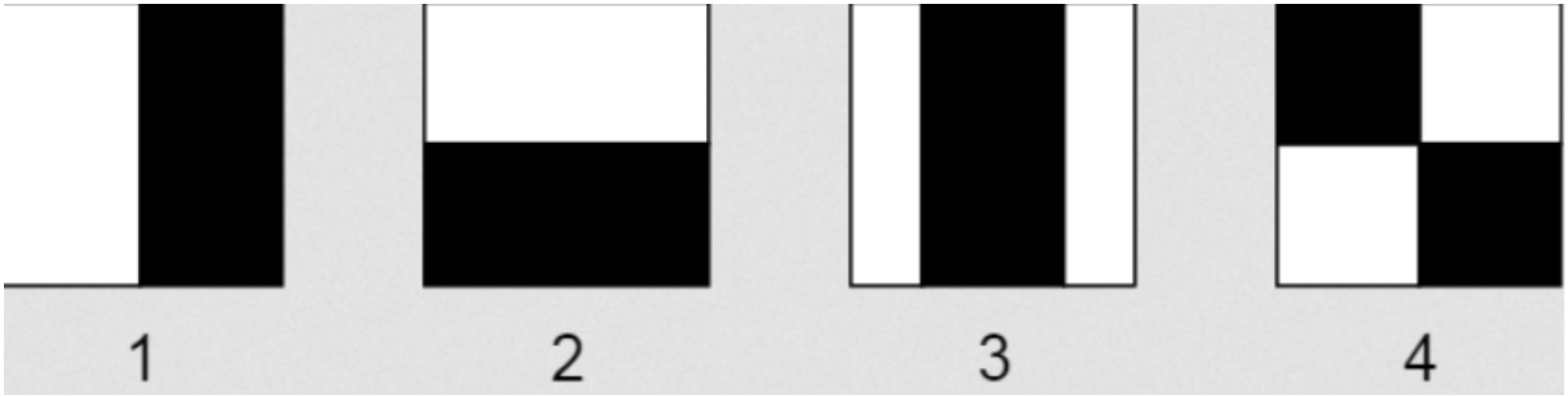
- nose
- eyes
- etc

As humans, we use those features to recognise other individuals

Then why wouldn't a machine use them as well?!

One of the first and most famous frameworks for face detection was presented in 2001 by Paul Viola and Michael Jones, often referred to as the Viola-Jones (https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework) object detection framework

It uses something called Haar-like features (https://realpython.com/traditional-face-detection-python/) to detect edges (1 and 2), lines (3) and diagonals (4)



It turns out that when overlapping these features in a human face, it can help us detect areas of interest

For example, *mask* (2) can help us identify the area with the eyes

Conversely, *mask* (3) can help us find the nose

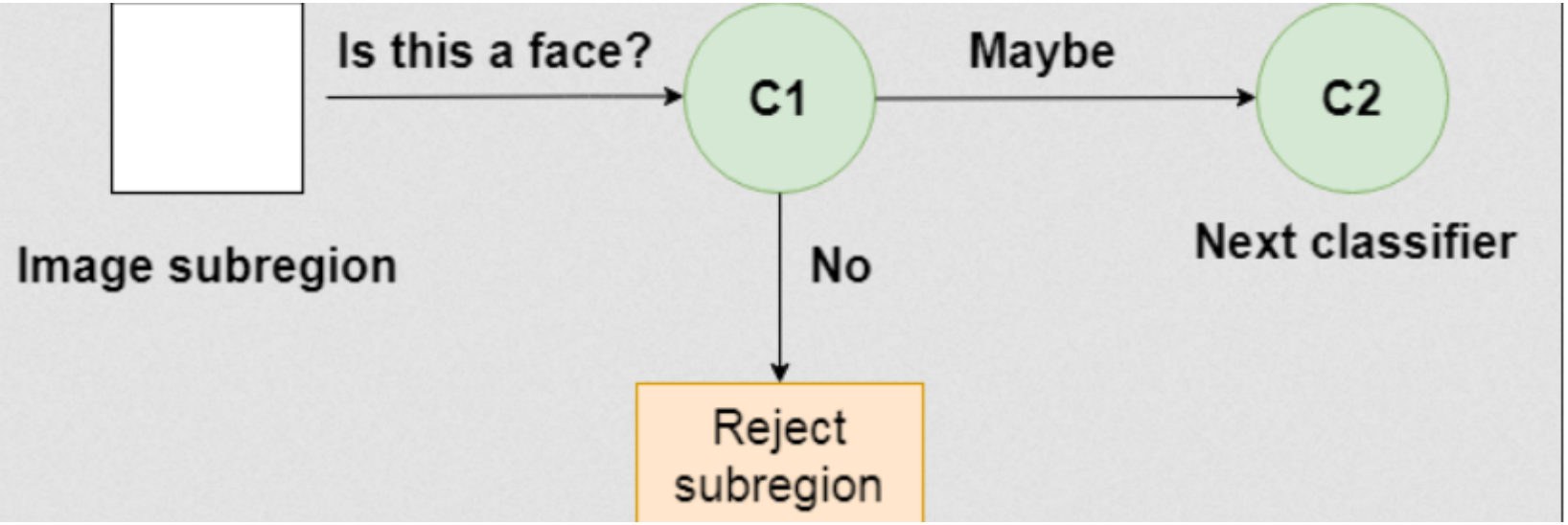
A simple classifier would right away deduct that, if there are eyes and nose, then there must be a face!

This is the algorithm most commonly used in commercial cameras (it is fast and easy to use)

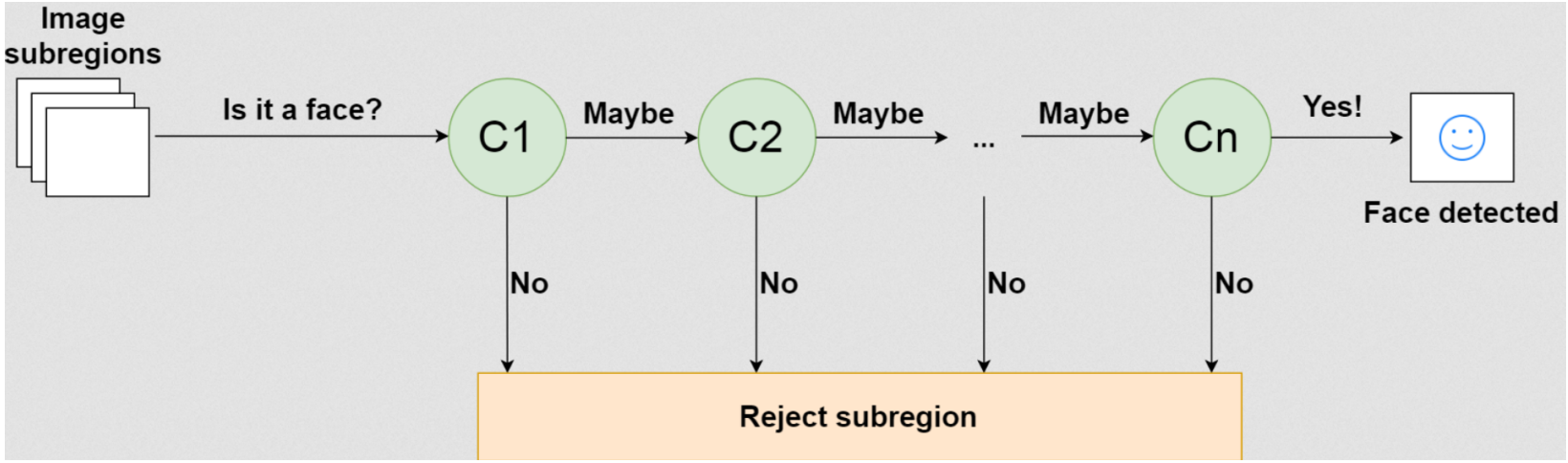
It is also why faces in statues and paintings get recognised!

It would take me a whole module to explain all the details of this method, the only thing you need to know is that it is an **end-to-end one**, this means that authors not only proposed the feature extraction, but also the classifier!

In this case, they use a **cascading classifier** (you will sometimes find this in literature as Haar cascade)



This can get more complex (but also more robust) as more classifiers are used

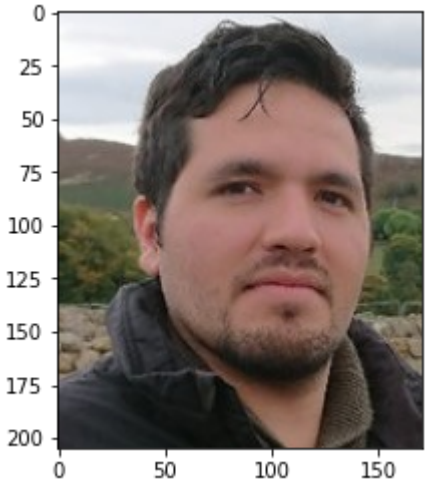


OpenCV comes with a ready to use Haar cascade!

First, we need to import an image

In [10]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
# Read image
original_image = cv2.imread('data/nomask.jpg')
# show converted because cv uses BGR, not RGB
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
```

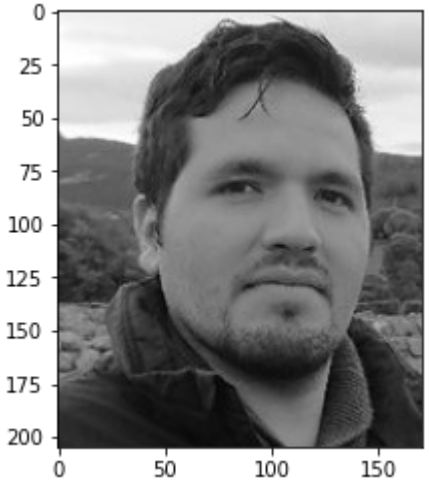
Out[10]:
```
<matplotlib.image.AxesImage at 0x1b07f60ccf8>
```



Then, we convert the image to grayscale so that we can apply the Viola-Jones method

In [11]:
```
# Convert color image to grayscale for Viola-Jones
grayscale_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
# We use "gray" to specify to plt.imshow that the image is grayscale
plt.imshow(grayscale_image, 'gray')
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1b07f68a978>
```



Then, we download the `frontalface.xml` model from [Github (https://github.com/opencv/opencv/tree/master/data/haarcascades)](https://github.com/opencv/opencv/tree/master/data/haarcascades), load it here and use it to process the image using the `detectMultiScale` function

In [12]:
```
# Load the classifier and create a cascade object for face detection
face_cascade = cv2.CascadeClassifier('data/haarcascade_frontalface_alt.xml')
# Detect faces
detected_faces = face_cascade.detectMultiScale(grayscale_image)
print(detected_faces)
```

```
[[ 54  43 107 107]]
```

Notice that we get four numbers! These are the coordinates where the face is found.

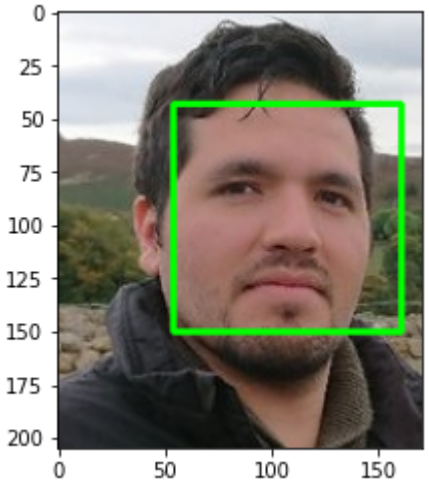
We can do a method to "draw" green rectangles over the faces as follows:

In [13]:
```
# Put rectangles in the images
for (column, row, width, height) in detected_faces:
    cv2.rectangle(original_image,(column, row),
        (column + width, row + height),
        (0, 255, 0),2)
```

Finally, we show the original image with the rectangle

In [14]:
```
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x1b07ffa9588>
```

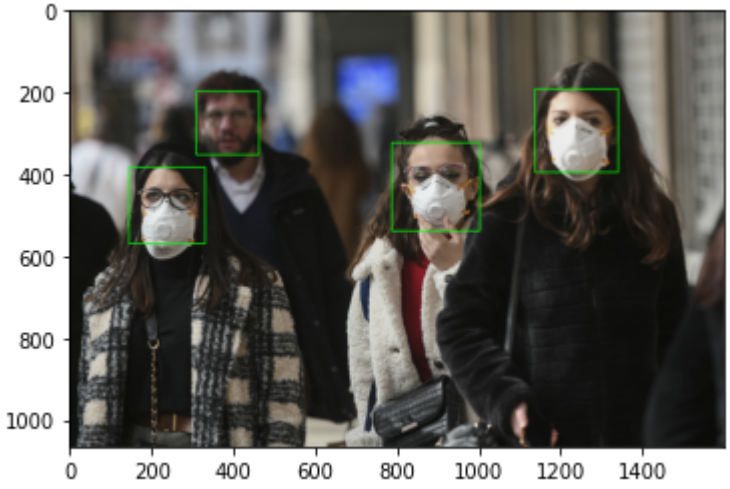


Do you think this works with a facemask?

In [15]:
```
original_image = cv2.imread('data/facemask.jpg')
grayscale_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier('data/haarcascade_frontalface_alt.xml')
detected_faces = face_cascade.detectMultiScale(grayscale_image)
print(detected_faces)
for (column, row, width, height) in detected_faces:
    cv2.rectangle(original_image,(column, row),
        (column + width, row + height),
        (0, 255, 0),2)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
```

```
[[ 146  383  186  186]
 [ 310  198  154  154]
 [1138  192  203  203]
 [ 788  323  215  215]]
```

Out[15]: <matplotlib.image.AxesImage at 0x1b00328cd68>



How about now?

In [16]:
```
original_image = cv2.imread('data/facemask2.jpg')
grayscale_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier('data/haarcascade_frontalface_alt.xml')
detected_faces = face_cascade.detectMultiScale(grayscale_image)
print(detected_faces)
for (column, row, width, height) in detected_faces:
    cv2.rectangle(original_image,(column, row),
        (column + width, row + height),
        (0, 255, 0),2)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
```

```
()
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x1b07f6375c0>
```

LAB 5: "FEATURE" BASED DATA CLASSIFICATION