Images/video basics & Object detection

Corentin Domken

Contact: corentin.domken@henallux.be

Image Basics

Grayscale Images

- Before we talk about how to use manipulate image files, let’s first discuss how computers handle images.

- Let’s imagine we wanted to build software that could sort out mails based on the zip code of the address.

Grayscale Images

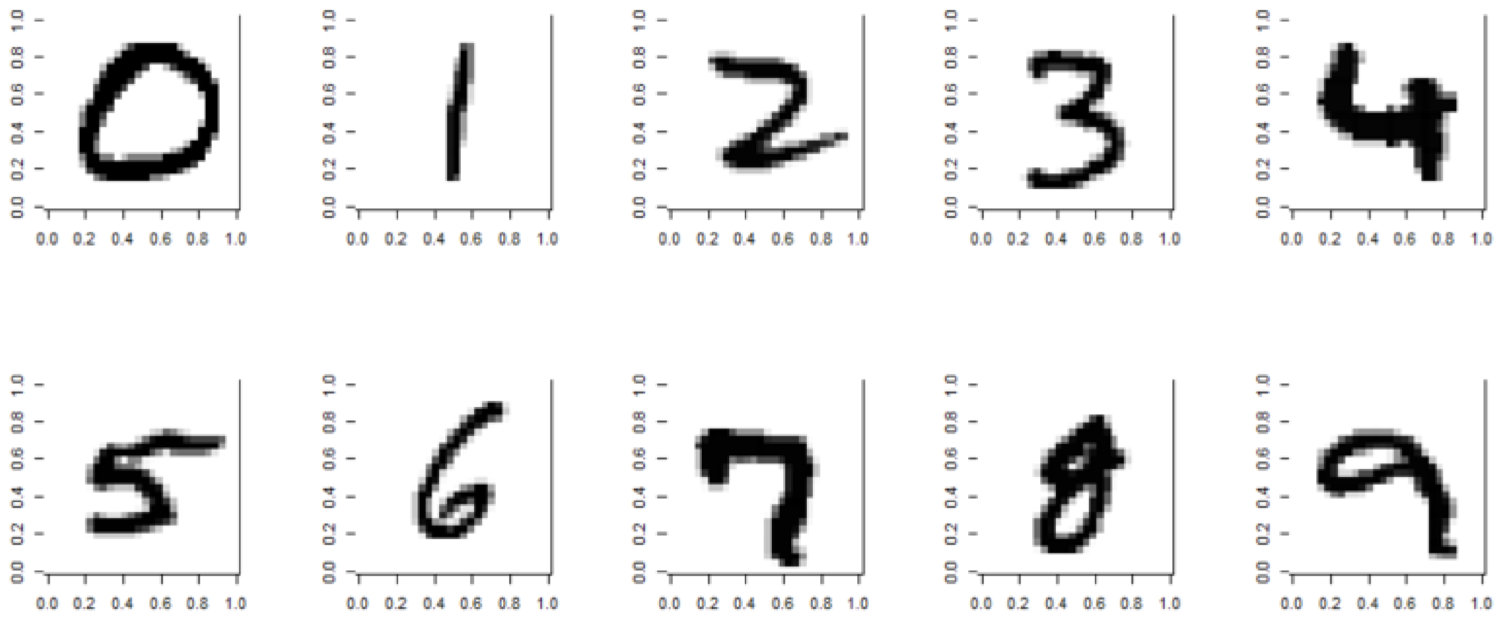
- Often mail is handwritten

  => we would need to begin to understand how to use computer vision to actual read image data of these handwritten numbers.

- How does a computer represent image data?
- Let's imagine we have a simple image of a handwritten number:

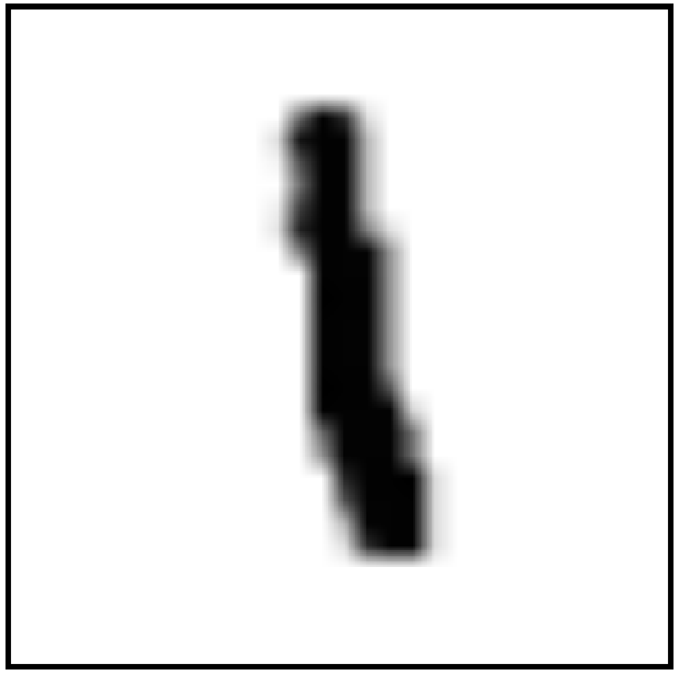
Grayscale Images

- Each single digit image can be represented as an array



Grayscale Images

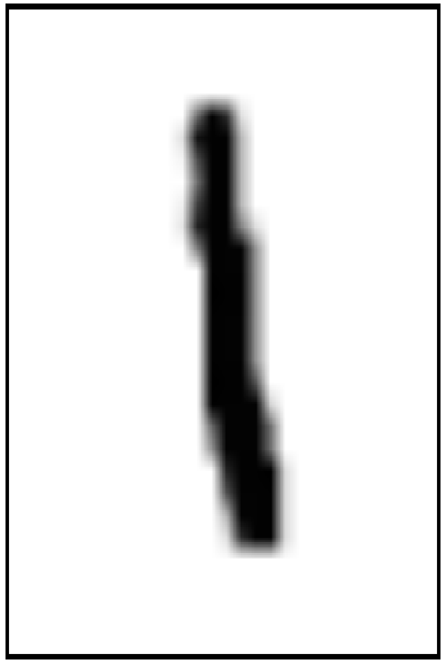
- For example, here a number is 28 by 28 pixels



$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

Grayscale Images

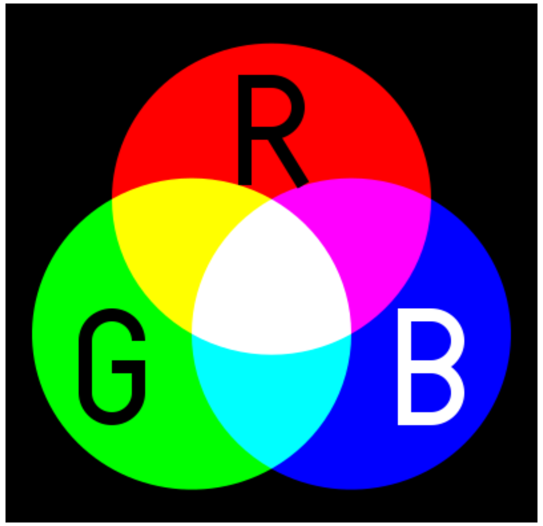
- Often the default images have values between 0 and 255 (The range 0 to 255 has to do with how computers store 8-bit numbers.)
- But you can always divide all the values by 255 to normalize between 0 and 1
- Then how dark a pixel should be can be represented as a value between 0 and 1



$$\cong \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

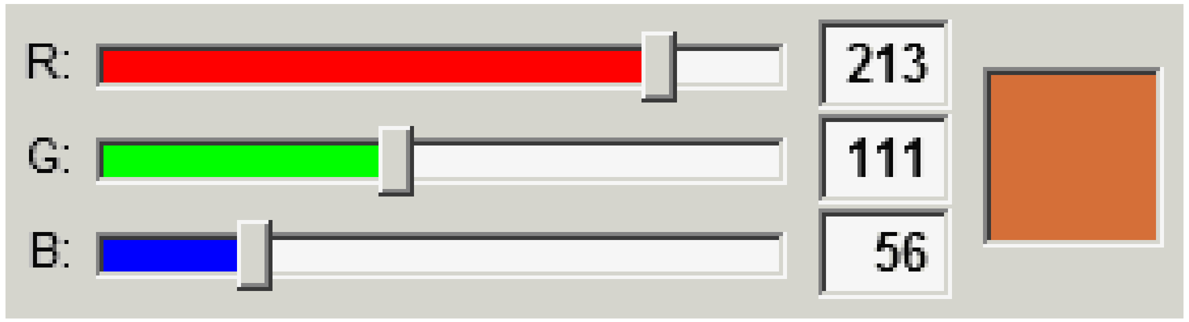
Color Images

- Color images can be represented as a combination of Red, Green, and Blue.
- Additive color mixing allows us to represent a wide variety of colors by simply combining different amounts or R, G, B.



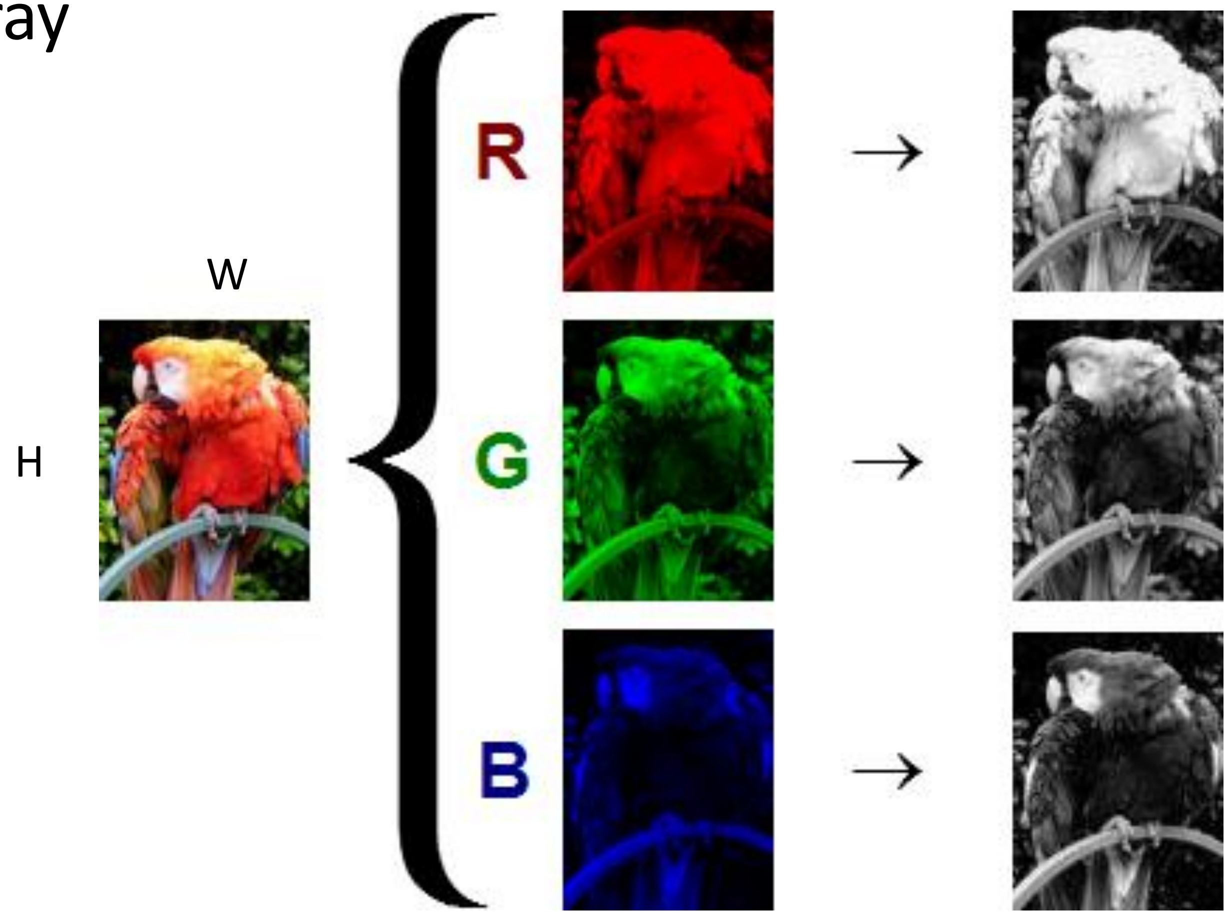
Color Images

- Each color channel will have intensity values
- You may have already seen this sort of representation in other software with RGB sliders



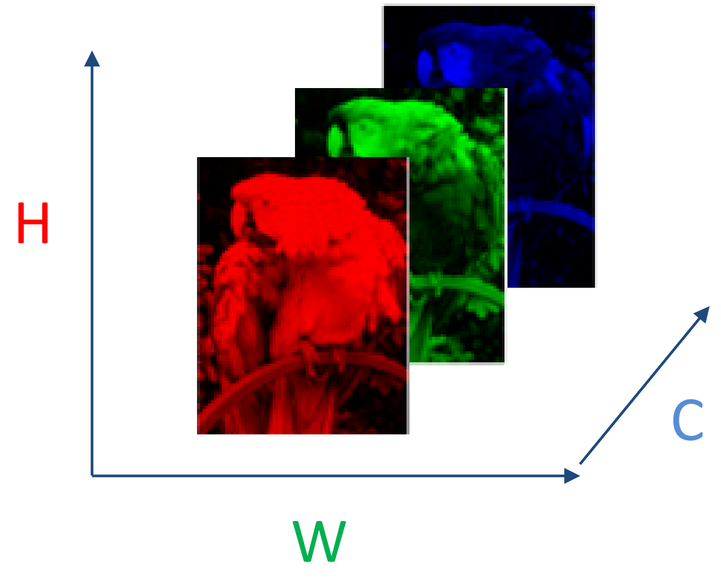
Color Images

- The shape of the color array then has 3 dimensions.
  - H: Height
  - W : Width
  - C : Color Channels



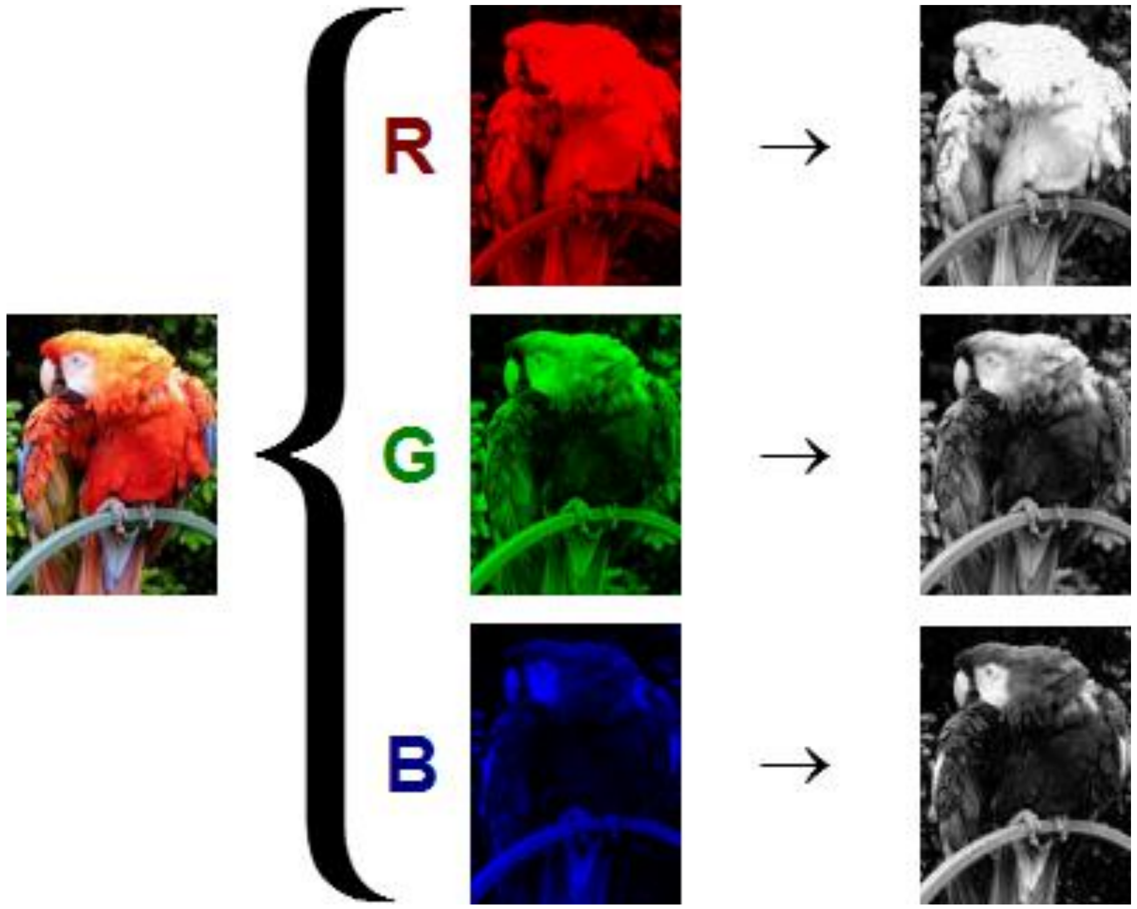
Color Images

- This means when you read in an image and check its shape, it will look something like:
- (720,1280,3)
- 1280 pixels width
- 720 pixels height
- 3 color channels



Color Images

- The user needs to specify which channel is for which color.
- Each channel alone is essentially the same as a grayscale image.

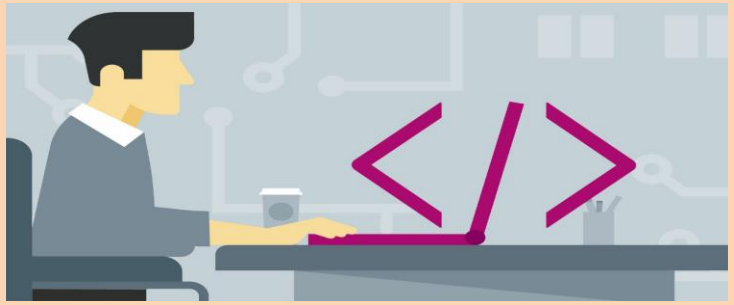


Color Images: More informations

- Please check out the Wikipedia article on RGB color channeling for more interesting details
- Nice youtube video: https://youtu.be/pmY7pOQCOr8

OpenCV for Images

- OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.
- Created by Intel in 1999, it is written in C++ (we will be using the Python bindings)
- It contains many popular algorithms for computer vision, including object detection and tracking algorithms built-in.



Reference notebook

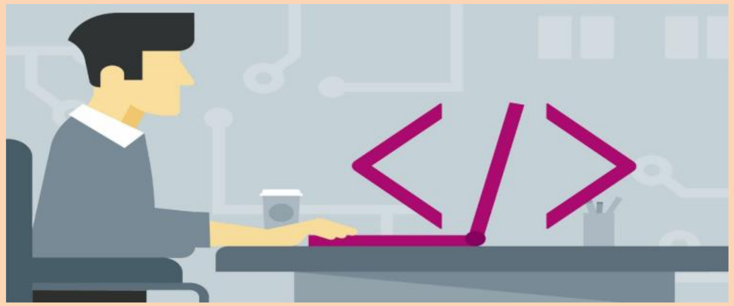
01-Image_basics.ipynb

- Section Goals:
  - Be able to open image files with OpenCV in both a notebook and a python script.
  - Draw simple geometries on images.
  - Directly interact with an image through callbacks.

Video Basics

Goals

- Goals are:
  - Connect OpenCV to a WebCam
  - Use OpenCV to open a video file
  - Draw shapes on video
  - Interact with video
- You will need a **webcam** to follow along this part!



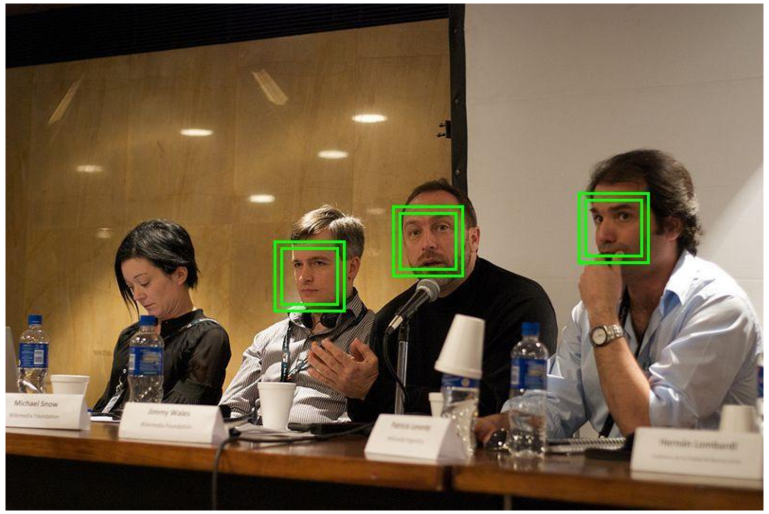
Reference notebook

02_Video Basics.ipynb

Object detection

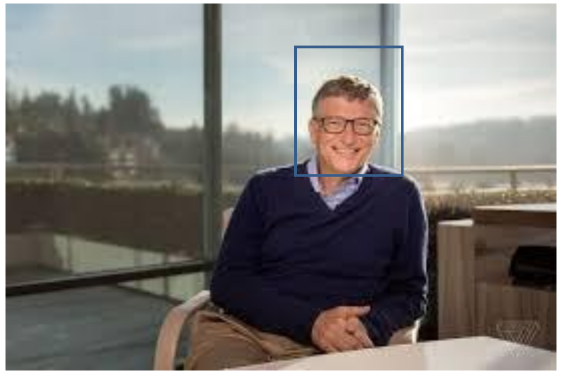
Face detection

- face detection :
  - Using Haar Cascades
  - detect if a face is in an image and locate it
- Keep in mind, this is face detection NOT face recognition.



Problem definition

- Facial detection : Where is the face?



- Face recognition: Who is this?

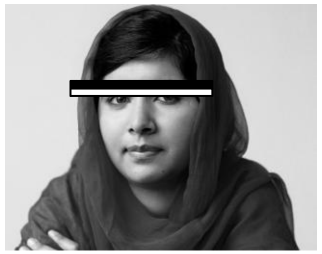
Haar Cascade Classifiers

- What is it?
- An object detection method that inputs Haar features into a series of classifiers (cascade) to identify objects in an image.
- They are trained to identify one type of object! (but can be used in parallel)

Concept

- HAAR classifiers are trained using a lot of positive images and negative images.

- We then extract features using sliding windows of rectangular blocks. These features are single valued and are calculated by subtracting the sum of pixel intensities under white rectangles from the black rectangles.



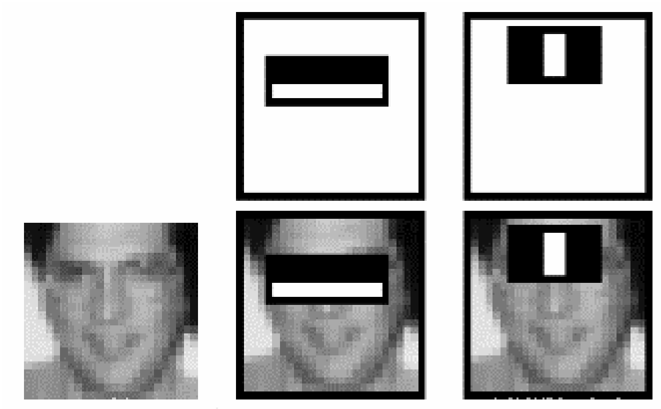
Face detection

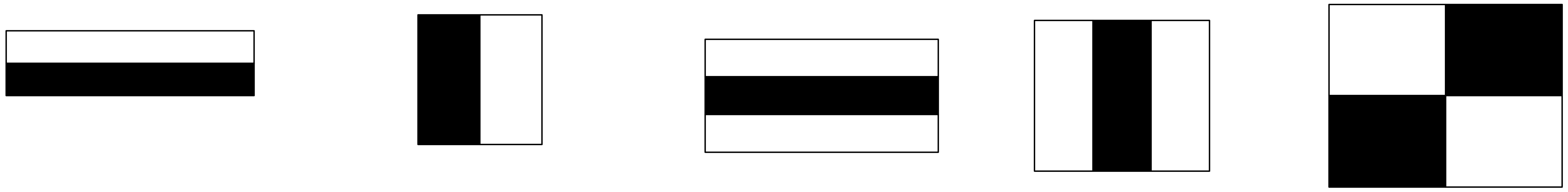
- In 2001 Paul Viola and Michael Jones published their method of face detection based on the simple concept of a few key features.

- Let’s first understand the main feature types that Viola and Jones proposed.

Haar Classifiers

- The main feature types are:
  - Edge Features
  - Line Features
  - Four-rectangle features





Face detection

- Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.
- Realistically, our images won't be perfect edges or lines.
- These features are calculated by:
    - **mean(dark region) - mean(light region)**

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

0	0.1	0.8	1
0.3	0.1	0.7	0.8
0.1	0.2	0.8	0.8
0.2	0.2	0.8	0.8

Face detection

- A perfect edge would result in a value of one.

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

0	0.1	0.8	1
0.3	0.1	0.7	0.8
0.1	0.2	0.8	0.8
0.2	0.2	0.8	0.8

Face detection

- These features are calculated by:
  - **mean(dark region) - mean(light region)**

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

0	0.1	0.8	1
0.3	0.1	0.7	0.8
0.1	0.2	0.8	0.8
0.2	0.2	0.8	0.8

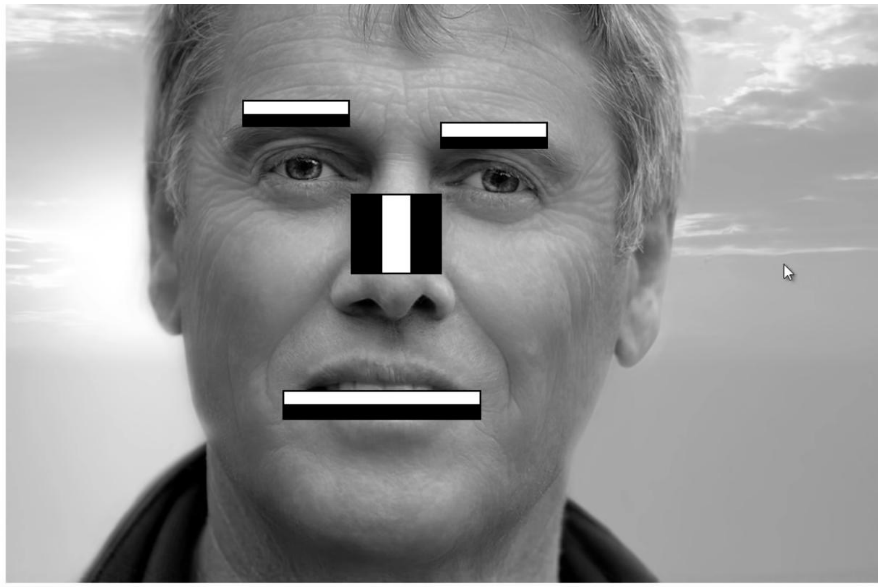
sumdark([0.8,1,0.7,0.8,0.8,0.8,0.8,0.8]) = 6.5
sumlight([0,0.1,0.3,0.1,0.1,0.2,0.2,0.2]) = 1.2
Meandark = 6.5 / 8 = 0.8125
Meanlight = 1.2 / 8 = 0.15
Delta = 0.8125 - 0.15 = 0.6625

Face detection

- The algorithm saves time by going through a cascade of classifiers.
- This means we will treat the image to a series (a cascade) of classifiers based on the simple features shown earlier.
- Once an image fails a classifier, we can stop attempting to detect a face.

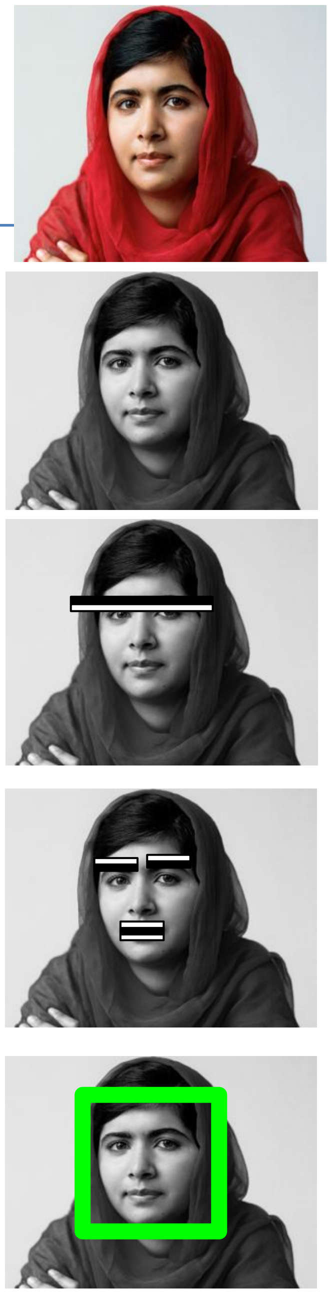
Face detection

- A common misconception behind face detection with this algorithm is that the algorithm slowly “scans” the entire image looking for a face.
- This would be very inefficient, instead we pass the cascade of classifiers.



Face detection

- First we need a front facing image of a person's face.
- Then turn it to grayscale.
- Then we will begin the search for Haar Cascade Features.
- One of the very first features searched for is an edge feature indicating eyes and cheeks.
  - If the image were to fail for the search of this feature, we can quickly say there is no face.
  - If it passed, then we search for the next feature, such as the bridge of the nose
- We continue through this cascade (which can be thousands of features).
- Until the algorithm detects the face.



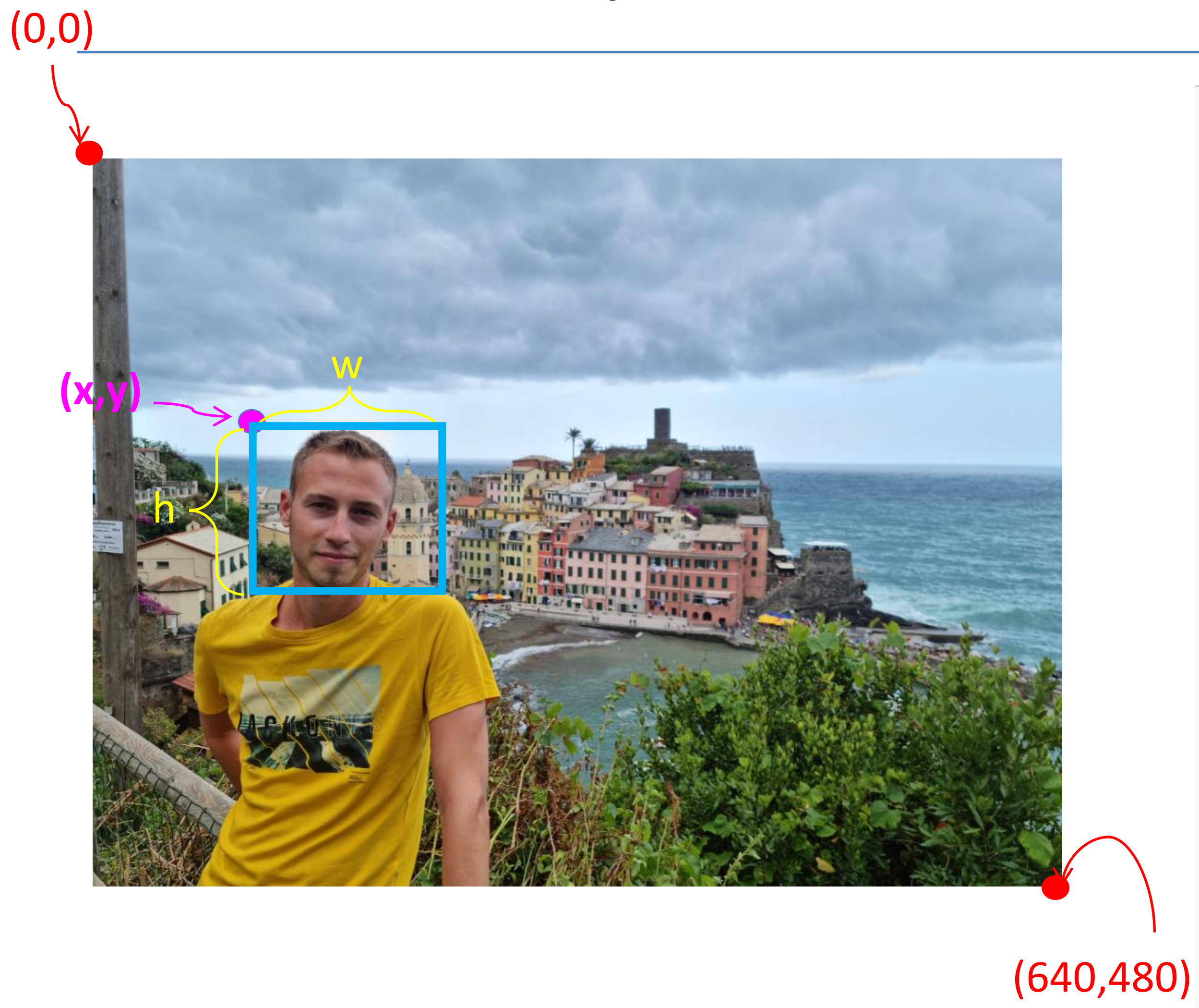
Object detection

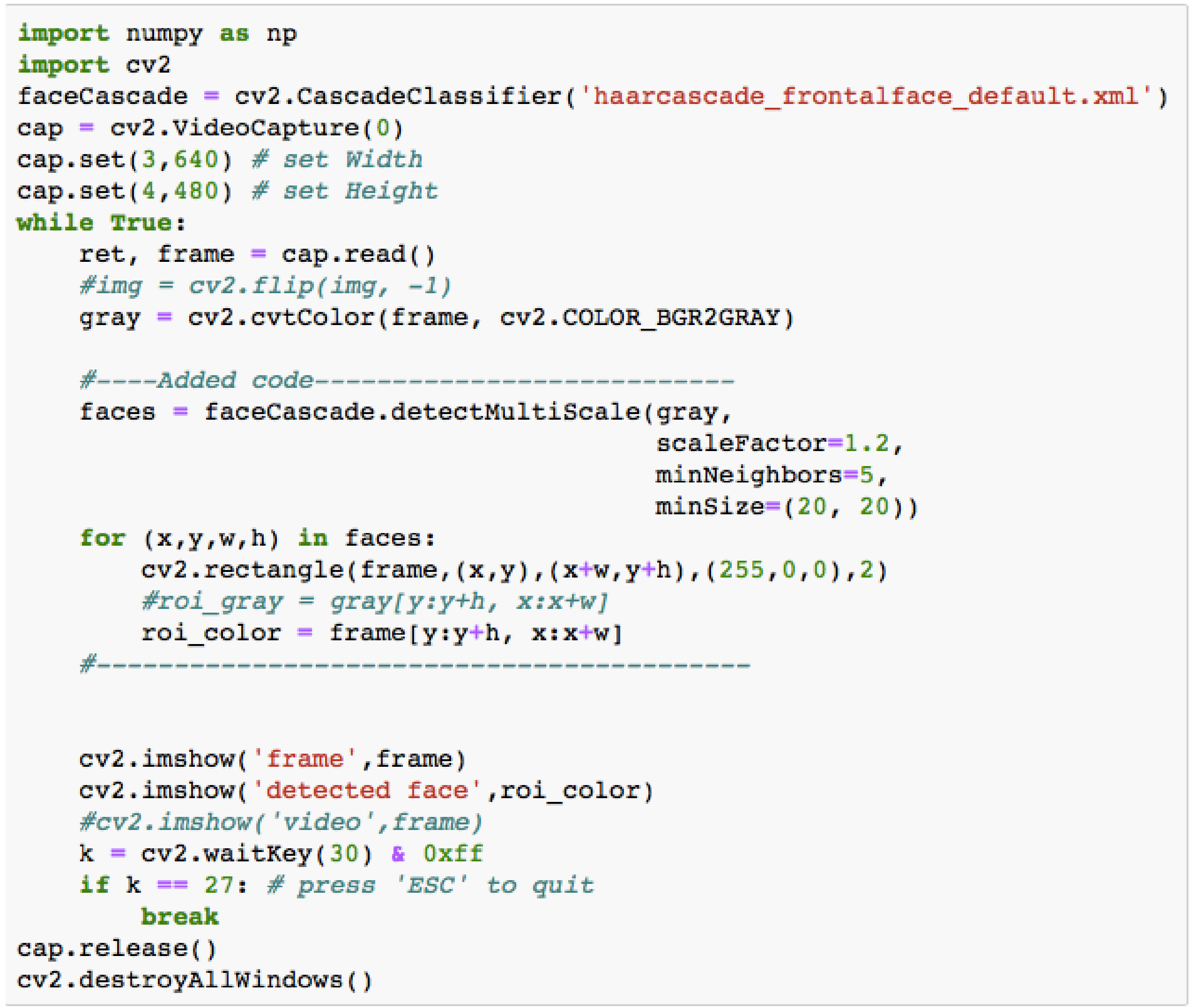
- Theoretically this approach can be used for a variety of objects or detections.
- For example we'll also work with a pre-trained eye detector.

Object detection

- The downside to this algorithm is the very large data sets needed to create your own features.
- However, luckily many pre-trained sets of features already exist.
- OpenCV comes with pre-trained xml files of various Haar Cascades.

Code example



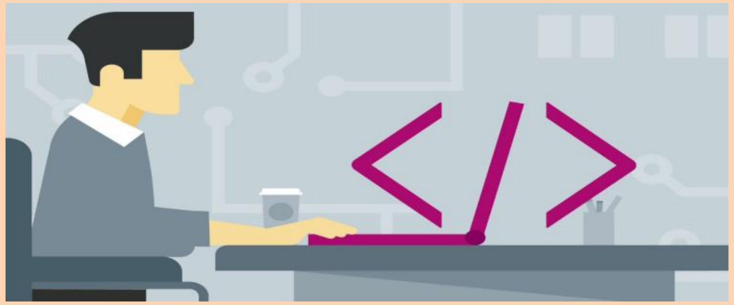


```
import numpy as np
import cv2
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Height
while True:
    ret, frame = cap.read()
    #img = cv2.flip(img, -1)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #----Added code---------------------------
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.2,
                                         minNeighbors=5,
                                         minSize=(20, 20))
    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        #roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
    #-------------------------------------------


    cv2.imshow('frame',frame)
    cv2.imshow('detected face',roi_color)
    #cv2.imshow('video',frame)
    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        break
cap.release()
cv2.destroyAllWindows()
```

Henallux-Engineering school Pierrard



Reference notebook

03-Face-Detection.ipynb
04-Detection-Assessment.ipynb

Questions

- https://www.youtube.com/watch?v=x41KFOFGnUE