February 03, 2023

Zuccolo Giada, *matr.* 2055702

DIGITAL FORENSICS PROJECT N.2

Stranger Faces

Introduction: Face detection and Face Recognition

Face detection is a technology that detect and identifies human faces in digital images, and it is used for biometric verification, to reduce crime and prevent violence.
Face detection is based on computer vision techniques to advances in machine learning (ML) to increasingly sophisticated neural networks and related technologies: they have the task to find human aspects on a video/image (for example it can start by searching for human eyes, eyebrows, the mouth, nose, etc.).
These algorithms work with "*feature extraction*" methods: extract salient information that is useful for distinguishing faces of different individuals and is preferably robust with respect to the geometric and photometric variations.
To help ensure accuracy, the algorithms need to be trained on large data sets incorporating hundreds of thousands of positive and negative images. The training improves the algorithms' ability to determine whether there are faces in an image and where they are.
Face detection is often confused with face recognition, but a **face recognition** system is a technology that can identify and compare a person's identity, determine if the face in two images belongs to the same person, or simply recognize a face from a digital image or face database. It uses biometrics to map face features that are unique to an individual in order to identify and match a person's identity from a digital image or face database. In fact, face recognition algorithms work by extracting some particular face features and measuring them in such a way as to identify matches with an already known face (typically within a dataset). Biometric security systems use face recognition to uniquely identify users during the authentication process.
SUMMARIZING, FACE RECOGNITION EXTENDS THE MEANING OF FACE DETECTION: IT NOT ONLY REVEALS THE PRESENCE OF A FACE, BUT ALSO DETERMINES WHOSE FACE IT IS.

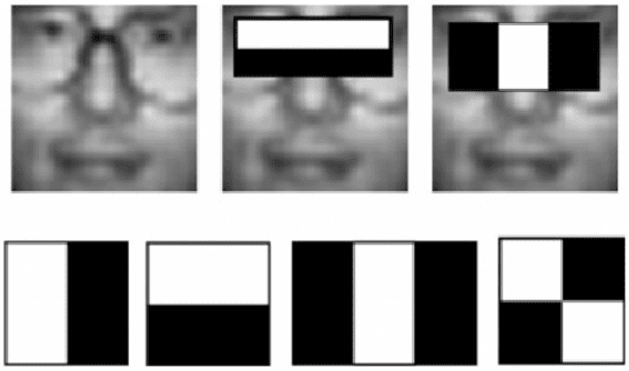
Setting the goals of the project

The goal of this project is to develop a tool capable of detecting faces in an image and video and then being able to recognize their identity (face recognition).
As a case study, it was decided to apply the face detection and recognition of three actors who are part of the famous TV series called "*Stranger Things*", both in a simple photo and in a video of their interview.
To evaluate the results obtained, general metrics will be observed, dictated by the algorithm that will be developed, by the frameworks used, and by the choice of these images and videos in relation to the dataset that has been created.

Theoretical Hints

After defining some concepts about face detection and recognition, let's go into more detail on a theoretical level. Obviously there may be more ways to develop an algorithm of this type, also because today there are many frameworks, libraries and CNNs that allows it.
So now let's discuss about the choices of technologies used for this project.
These tools are written in **Python** and it will use the potential of the **OpenCV library**, which already implements one of the most used classification algorithms in this context, known as **Haar-Cascade**.
In fact, face detection (and recognition) is basically a *classification problem*, that is, a problem of distinguishing between different classes and to recognize the human face, it is necessary to detect a face then extract the face features. Many methods have been created and developed in order to perform face detection: one of the most popular methods is a classification algorithm **Viola-Jones Haar Cascade Classifier**. It can detect objects in images, regardless of their scale in the image and location. It is a machine learning method that trains a multilevel function using as many images as possible with and without an object to detect: initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier.

In simple words, Viola-Jones Haar Cascade Classifier method calculate Integral Image through Haar-like feature with AdaBoost process to make a robust cascade classifier. Like we said, it uses Haar features (that are like convolutional kernels) to *extract features*.

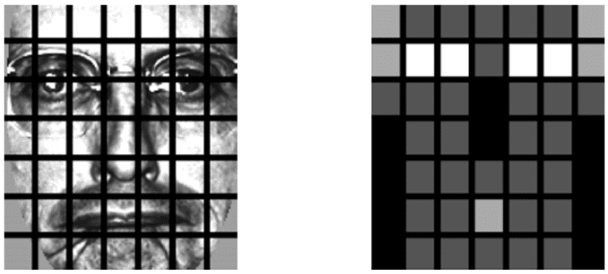


As you casn see on the image above, a *Haar-like feature* is represented by taking a rectangular part of an image and dividing that rectangle into multiple parts. They are often visualized as black and white adjacent rectangles. With these types of haar like features, we can *extract features*: each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.
Various variations of these regions of different sizes are convolved through the image in order to get multiple filters that will be inputs to the AdaBoost training algorithm.
In simple terms, each feature acts as a binary classifier in a cascade filter. If an extracted feature from the image is passed through the classifier and it predicts that the image consists of that feature then it is passed on to the next classifier for next feature existence check otherwise it is discarded and next image is checked.

OpenCV already contains many pre-trained Haar Cascade classifiers for face, eyes, smile etc, stored in XML files. For this project, it will be use haarcascade_frontalface_default.xml

Note: actually, in the experiments, CNN gives better detection accuracy than Haar Cascade. However, for the client application face detection in some specific devices, haar cascade can be used wider than CNN face detection. Haar Cascade is still useful, particularly when working in resource-constrained devices when we cannot afford to use more computationally expensive object detectors, exactly the case of this project.

What about the face recognition? With the facial images already extracted, cropped, resized and usually converted to grayscale, the face recognition algorithm is responsible for finding characteristics which best describe the image.
There are different types of face recognition algorithms. For this project, it will be used **Local Binary Patterns Histograms (LBPH)**, which is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.
Given a face in a dataset, the first step of the algorithm is to divide the face into 7×7 equally sized cells, and for each of these cells, we compute a histogram. In this way, we actually are able to encode a level of spatial information such as the eyes, nose, mouth, etc., that we would otherwise not have. This spatial encoding also allows us to weigh the resulting histograms from each of the cells differently, giving more discriminative power to more distinguishing features of the face, as you can see on the image:



It's important to note that the LBPs for face recognition algorithm has the added benefit of being updatable as new faces are introduced to the dataset.

Development

After having thoroughly explained the theory and the technologies that we have chosen to use, let's move on to the development part.

The dataset

The dataset was the cornerstone of the project.
As there was no useful dataset for this project, all the images in this dataset were downloaded manually, with the aim of having as many images and different points of view as possible and above all not repeating images. Since these images were downloaded manually, they were all different sizes and angles. Therefore, a small tool was also developed to clean up this dataset, implementing the classification algorithm to recognize the face in the image and to save it in the appropriate directory, specifying the dimensions. This allowed to have more accurate results in the next steps.

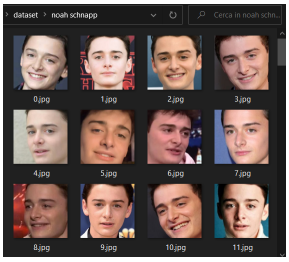


Figure 0.1:
A small glimpse of
Noah Schnapp's dataset

There is a consideration to be made (which will be extensively covered in the next paragraphs): the objective was primarily to have a balanced dataset (objective achieved), but it was not possible to also have a dataset rich in images. This is a weakness of this project, which, despite this point, has led to good results in terms of accuracy.

A file labels.pickle will contains the labels, that are the names of the directories (that are the names of the three actors: "finn-wolfhard", "millie-bobby-brown" and "noah-schnapp").

Create trainer.py and trainer.yml

The file called "trainer.py" contains the tools that train the algorithm for the recognition of the faces. In particular, it creates the *labels* that will be identified by a numeric value (id_), and saves them in a file called "labels.pickle". With these, it is able to assign to each face it detects, the name of the person it recognizes.
At this point, the algorithm goes through each folder of the image dataset, and for each of them it checks each image and detects each face, in order to obtain a data file called "trainer.yml". This is not a viewable or openable file, but it is a file that is examined each time the tools for facial recognition in the images or video are run. Inside, it contains all the data that connect each label to the set of features of the face belonging to the given person. We can therefore say that the "trainer.py" file takes all the images of the dataset and transforms them into trainable data, taking pixel by pixel and transforming them into numerical data. Of course it will use for training it will take the pixels of the image referring to the face it has detected.

The tools: faces_img and faces_video

In order to implement the script, the first thing to do is import OpenCV (represented by the cv2 module) and load the file containing the data of the classifier, with the instruction: face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml'). The image, on which the facial recognition will have to perform, will be opened and converted to grayscale (the classification we are using works exclusively on grayscale images).

Next, we need to use the classifier by applying the function:
face_cascade.detectMultiScale(grayed_image, scaleFactor, minNeighbors)

The "scaleFactor" parameter is required by the Haar-Cascade classifier to make a series of scaled-down versions of the original image, forming a sort of leveled pyramid.
it is a crucial representation for the classifier, since the process of extracting the most salient features of the image is based on it, then used for the actual recognition
It basically controls how the input image is scaled before detection, such as whether it's zoomed in or out, which can help you find faces in the image better The default value is 1.1 (10% increase), although it can be reduced to values such as 1.05 (5% increase) or increased to values such as 1.4 (40% increase). A value of 1.2 was applied for this project.

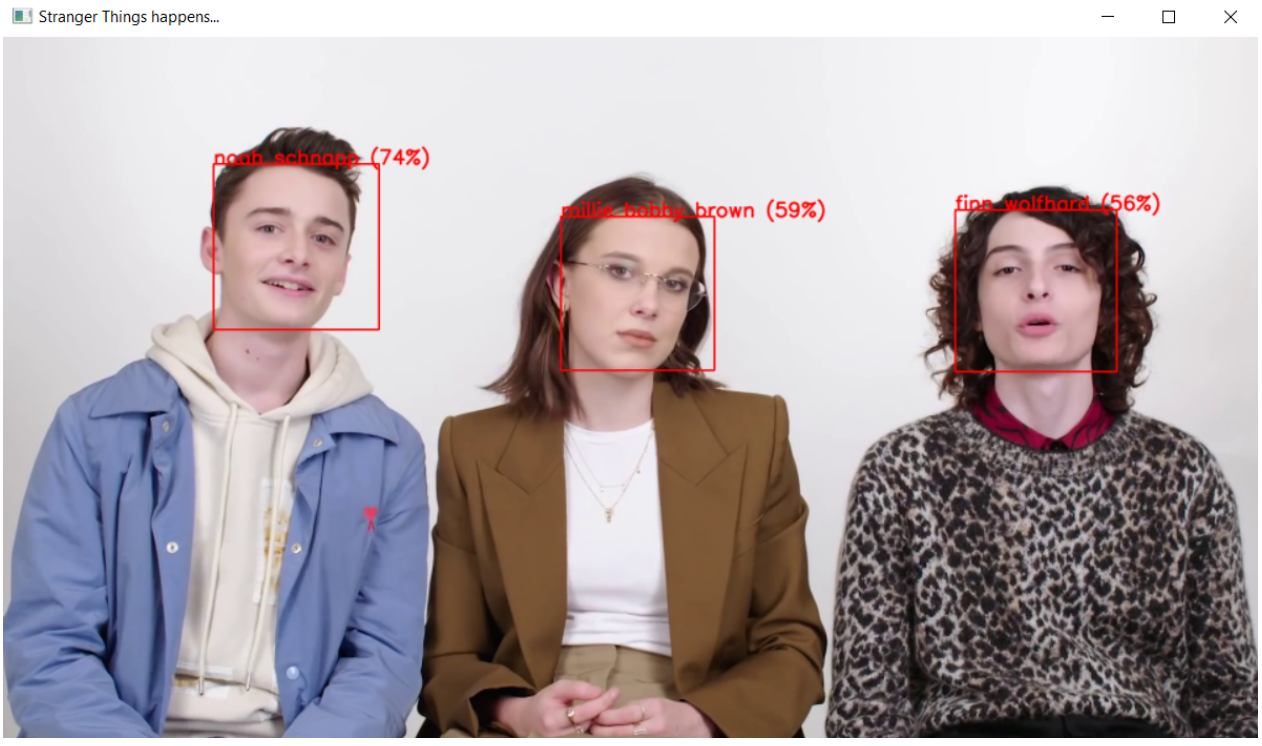
The "minNeighbors" parameter represents the number of nearby regions that must be considered by the algorithm for each of the candidate areas to be recognized as faces: it determines how robust each detection must be in order to be reported.
Higher values generally result in fewer recognized faces, but better recognition quality.
The default is 3, but can be reduced to 1 to detect many more faces and will probably increase false positives, or increase to 6 or more to require much more confidence before a face is detected (3 6 is a good value). A value of 5 was applied for this project.

As regards the tool that takes care of facial recognition in the images (called faces_img.py), two situations have been addressed: one photo is from an interview that the actors have released, the other photo instead portrays them in the role of the protagonists of the TV series, during a of the episodes.
It may seem trivial, but their aesthetics have been slightly modified for the series and therefore it is an interesting perspective to be able to see how the algorithm works.

As for the tool that deals with facial recognition in the video (called faces_video.py), the procedure is the same.
In fact, video is nothing more than an ordered and following collection of images (frames).
The tool was developed in such a way as to detect (and close in a square) the face and recognize the reference name (therefore the label), throughout the duration of the video.
A percentage of confidentiality is entered next to the name and is constantly updated. When this value turns out to be too low, instead of showing the relative label, it will show "unknown".
The video used portrays the actors during an interview they released on a youtube channel, therefore outside the film set (therefore their appearance is different from what they have in the TV series).

In the image below: on the left side we have the actors during an interview with the relative facial recognition, on the right side we have the actors on the set during a scene from the series with the relative facial recognition.

In the image below: a frame extracted while the facial recognition tool in the video was operating.



Conclusion with final evauations and results: discussions about the dataset, confidence value, detect parameters, and the changing ages

First of all, to perform the script at its best, a dataset full of images is necessary.
In this project, the number of collected images is certainly very low and just enough to have good results. An improvement to be adopted to better train the recognizer of this tool is to fill the dataset even more with images. If we further increased the number of datasets, then the **accuracy** will also increase, also considering that **LBP** system can detect the desired person with the accuracy of 89.1%.
So the value of **confidentiality** (that is how much the trainer is confident with the assigned label) also definitely increases.
In both tools this value is quite low, but it has been set so as to be able to show the name of the recognized face in the output, with the percentage next to it.

It should also be emphasized that the trainer used (frontalface_default) is a trainer indicated for videos/images in which the person's face is seen from the front: in the video in particular, the actors were very often shot in profile, and this is why the confidentiality value is lower. To avoid this hitch as much as possible, all the various possible combinations of the "scaleFactor" and "minNeighbors" parameters have been thoroughly studied.
After careful analysis and a very long test phase, the combination that led to better and more performing results in several situations (from the photo of the actors during the series, to the video of their interview off the set) includes the parameter "scaleFactor" with a value of "1.2" and the "minNeighbors" with a value of "5".
They are both very good values, which do not deviate from the theory behind the algorithm. However, it cannot be excluded that with a richer and more uniform dataset, better results can be achieved, and can be achieved with different values of these two parameters.

I had to specify "*uniform* dataset" because, unfortunately, on the web there are many "*discordant*" photos of the actors, due to the make-up/wig they have in different contexts: in the TV series they are very different from how they are in reality, as you can see from the image below, where on the left there are the actors during a scene from the series, while on the right a photo of them taken around the same time:

Furthermore, to further undermine the recognizer is the fact that the actors have become adults in recent years, therefore on the internet there are mainly photos where they grow physically and slightly change their physiognomy, as can be seen from these two shots below of the actor *Finn Wolfhard*, where the two photos were taken approximately 2/3 years apart (maximum).

However, in the facial recognition tool in the video (the tool with the most weaknesses), analysis was done on the basis of the frames.
For each label (actor) the number of frames in which it appears was counted and put in proportion (percentage) with the total number of frames shown.
Similarly, the same work was done with the times in which he was unable to recognize the face, and the word "unknown" appeared in the square.
Furthermore, for each actor, for each frame in which he has been recognized, the respective value of the confidentiality that he assumes in that particular frame is saved, so as to show a final total average.
Every time the tool is run, all these values are then shown in the output.
The values shown in this table are an example of the output obtained (the conf value was set as 56%).

actor (label)	**counted frame on total**	**mean confidentiality value**
finn-wolfhard	116/220 (52.73% of frame)	with a mean conf of 60%
millie-bobby-brown	116/220 (52.73% of frame)	with a mean conf of 65%
noah-schnapp	146/220 (66.36% of frame)	with a mean conf of 76%
unknown	32/220 (14.55%)	

So, in the end, despite the various weaknesses listed above, this data is quite satisfactory.
Especially emphasizing the fact that the video has moments in which there are only writings, therefore without the actors, and unfortunately those frames (which are still few) are counted in the total. So these data, although positive, are still more negative than the actual reality.