Digital Forensics A.Y. 2019/2020
Project 3:
Face Detection and Classification for video signals using Matlab, Python & OpenCV

Donald Shenaj
1238939

July 2, 2020

Contents

1 Introduction

For this project we were asked to build a face recognition strategy, starting from the lab experience n. 3. In particular I designed a video processing system which is able to detect and classify all the 13 actors of the dataset provided.

I decided to implement in matlab only the training of the neural network leaving the task of face detection and classification of their content completely in python exploiting the well known OpenCV libary.

In this way it is was possible to design a *main.py* which given in input a video displays in output graphically the labeled frames which happens to be the original video frames with the additional information found: bounding boxes over the detected faces and their predicted classes written above.

2 Images Dataset

3 Face detector

The first step needed in our system is the face detection, and to do so I decided to use the Viola and Jones algorithm which is one of the most used method for real-time applications of face recognition and localization, and we studied it on the Computer Vision course.

3.1 Viola and Jones algorithm

Paul Viola and Michael Jones in their paper [1] proposed an object detection method using Haar feature-based cascade classifiers and it consists of three main concepts:

- *Integral image*: it is a way to represent an image starting from the initial one with some operations which allows to compute filtering with different windows size at the same speed.
- *Adaboost*: it is a classifier which allows to select a reduced number of relevent features, discarding the ones which are not really significative.
- *Cascade classifiers*: it is realized by using a set of classifier with increasing complexity. Since in general the images have lets say a limited number of faces, it has to be fast to discard the non faces with the first stages of the classifiers, in this way the algorithms performs faster.

As we already mentioned this algorithm is very fast but of course is not perfect. In fact it is not rotation invariant so it will miss all the rotated faces and also the haar

features can get false positive when analizing regions with contraposition of white and black pixels. For instance a typical false positive detection happens when we feed an image of a drawn white and black face, which is not a real face. And also some other typical false positive are the body parts.

3.2 OpenCV implementation

Since the Viola and Jones algorithm is very used in OpenCV, we can import the model already trained by them in our code and use it immediatly. By running the algorithm over an image we get the bouding boxes of the detected faces as a set of coordinates, and to get a feedback we display those rectangles over the starting image. For instance in Figure 1 we can see that the one face is correctly detected because it is delimited by the green bouding box. Instead the other face is not possible to be detected with this algorithm, for the reasons previously mentioned.

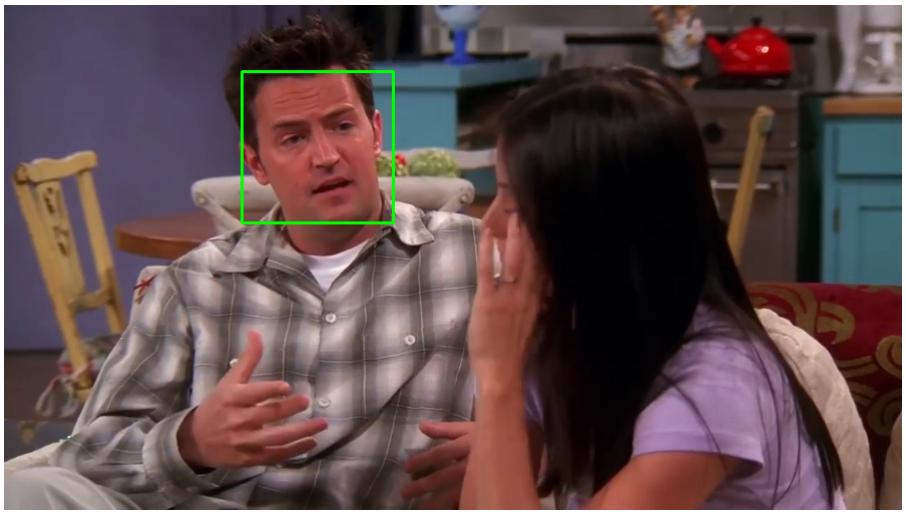

Figure 1: Example of image after applying the Viola and Jones face detector with OpenCV

4 Face classification

At this point we want to build a convolutional neural network which has to be able to classify the content of the bounding boxes previously detected. To do so I decided to work on Matlab and change the CNN model seen in the lab in order to get an higher accuracy for the 13 classes. Our goal is also to have a system which is able to process video in real time, so we don't want the network to be too slow and we'll need to make a trade-off between accuracy and speed of the processing.

4.1 CNN design

Starting from what we have seen in the example code for the lab 3, I changed some parameters to increase the performances. First of all I added one more convolutional layer, since the classification between 13 classes it's possibly harder the classification between 3 classes. But of course I avoided to build a network too complex like the AlexNet, because i can increase the accuracy of the system by simply considering consecutive frames without losing the fast processing.

After that I've added a additional dropout layers since I wanted the network to not depend too much on single neurons and overall reduce the overfitting.

Finally to increase the performance I've added additional filter channels in the convolutional layers, in this way it was able to extract more interesting features.

After carefully changing the parameters and watching how the training changes, I found the best result with the following architecture:

- Input layer receives input images with shape (64,64,3);
- First 2D convolutional layer with 64 filters of size (3,3), stride of (1,1), followed by a batch normalization, ReLu activation, and finally a max pooling layer with size 2 and stride 2;
- Second 2D convolutional layer with 128 filters of size (5,5), stride of (1,1), followed by a batch normalization, ReLu activation, and finally a max pooling layer with size 2 and stride 2;
- Third 2D convolutional layer with 128 filters of size (8,8), stride of (1,1), followed by a batch normalization, ReLu activation, and finally a max pooling layer with size 2 and stride 2;
- Forth 2D convolutional layer with 128 filters of size (9,9) padded to be at the same size as the input tensor, followed by a batch normalization, ReLu activation, max pooling layer with size 2 and stride 2, and finally a dropout layer with $P = 0.25$;
- Last mono dimensional dense layer, with softmax activation used for the classification.

4.2 Training

The model was then trained with a batch size of 128 samples in 6 epochs, obtaining an accuracy of 98.721704, which is very good result.

In Figure 2 we can see the training progess and in Figure 3 the confusion matrix.

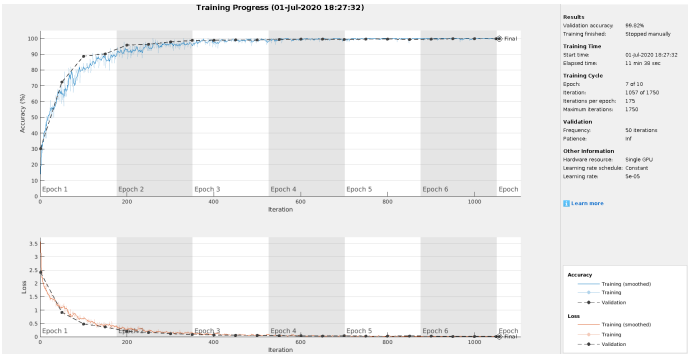

Figure 2: Training progress of the CNN

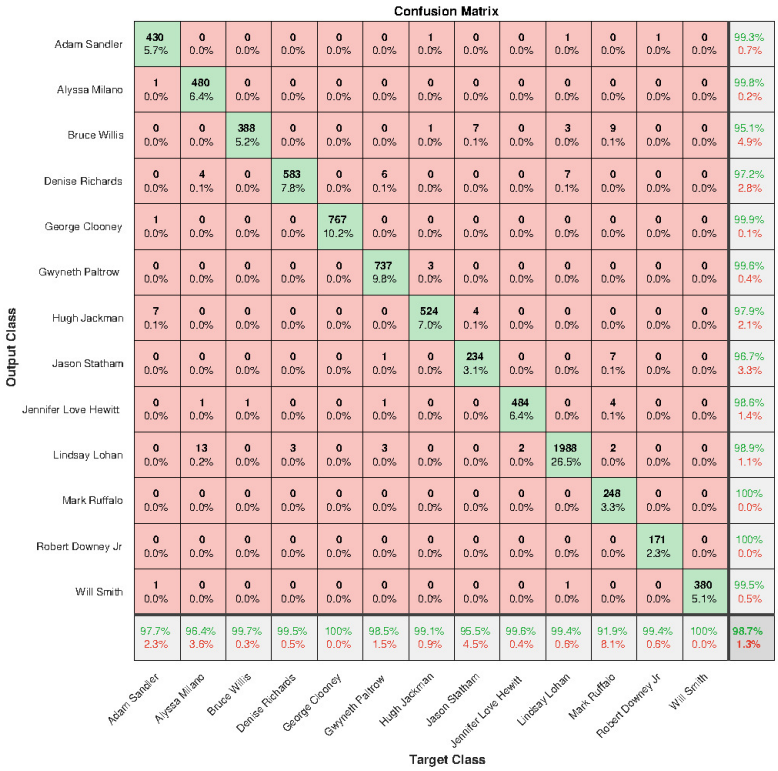

Figure 3: Confusion matrix of the CNN

5 Final system

At this point we have all the pieces needed to design the complete system. The implemented python codes are the following:

- *main.py*: Main code, receives in input a video and displays the video with bounding boxes and predicted labels;
- *video_utils.py*: Utility functions invoked by the main code;
  - *video_url()*: Mapping of actor names and youtube video links;
  - *video_classifier()*: Complete video processing, detection and classification;

5.1 Video processing

To process the video frames I have used the python libraries OpenCV and VidGear [2]. OpenCV was used in addition to what we saw for the detection, even for the classification by exploiting the DNN module. In fact after we train our network with Matlab it is possible to save the weights with the ONNX format and load that network with the method *cv2.dnn.readNetFromONNX()*. The trained network will be used to make predictions (classifications) over the new images (video frames) provided. Then we will display graphically the results of the processed video frames.

VidGear instead was used for extracting directly the video frames from any youtube video without the need to download them locally.

5.1.1 Video dataset

The videos I have used to test the system are all taken from youtube, in this way it is very easy to change them and test with more videos. But of course we can provide even non youtube videos if we have them stored in our computer. The only challenging part is to find videos where the only person we see is one of the 13 actors, since we want to use multiple consecutive frames for the face classification process.

The videos are one for each actor and are linked in the method *video_url()*.

5.1.2 Detection and classification

The function used to detect and classify is *video_classifier()* inside *video_utils.py*. It is firstly used the cascade classifier to extract the bouding boxes of the present faces, then it's extracted the region of the image, applied a filtering and finally it is feed to the input layer of the CNN and forwared to the output layer to get the

predicted class. This operation is done for N consecutive frames, to be choosen, and the most common result between the N consecutive classification is considered as the predicted class.

In this way we increase the accuracy of the prediction considering also the fact that all the false positive face detected are mosly discarded by the final classifier when they are not detected for N consecutive frames.

This is very nice but of course we are losing something. In fact in the possible scenario of multiple faces per frame it would have been necessary to use a video tracking algorithm in order to compare correctly the classification for the same face among N consecutive frames.

5.2 Main

Finally to run the code the instruction is simply the following, where instead of 'videoname' of course we have to set one of the youtube videos mapped in *video_url()*. This was though in order to have a more user friendly instruction.

```
python main --input videoname
```

We can see the result of final classification of Bruce Willis in Figure 4.

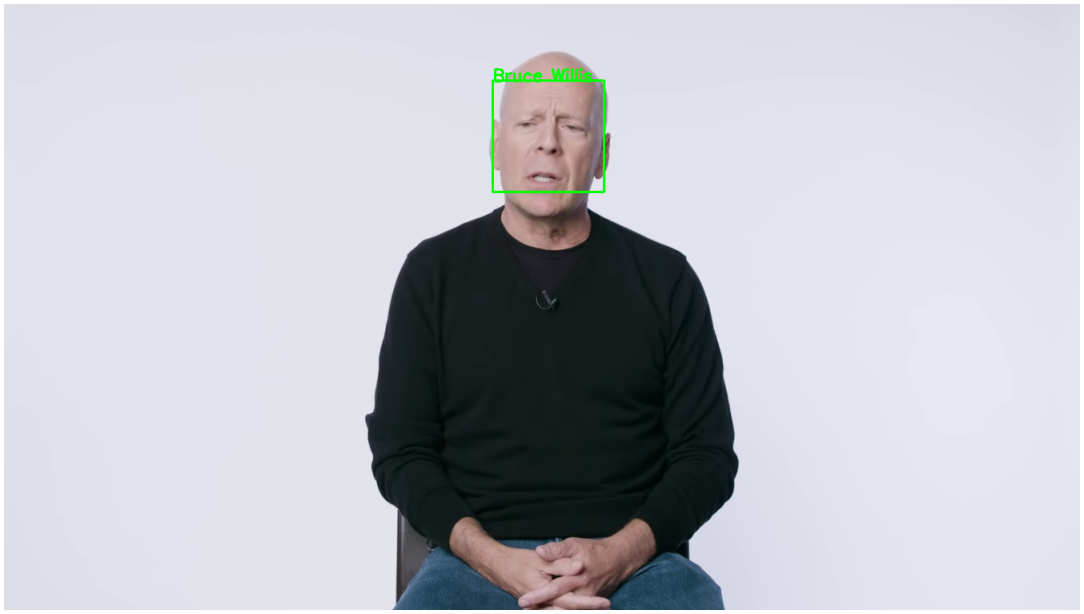

Figure 4: Result of the detection and classification of a Bruce Willis video.

6 Conclusions

In this work we saw that the designed system can be effective in the detection and classification of people faces even for real time applications if we can accept a lower accuracy. We have not analyzed the case of the presence of multiple faces along the different frames of a video since that would have required the use of some video tracking algorithms in general much more computational demanding. But of course we can still use our system for multiface detection and classification if we use just one single frame for the classification by setting $N = 1$, losing some accuracy.

References

[1] P. Viola, M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, International Conference on Computer Vision, 2001

[2] Abhishek Thakur: *vidgear* , `https://github.com/abhiTronix/vidgear`, 2019-2020