Face Detection

Digital Forensics Project Report

Giovanni Gallinaro

A.Y. 2019/2020

Abstract

Face detection and face recognition are widely used techniques for forensics and security purposes. There has been a huge improvement in this field in the recent years, with the introduction of machine learning strategies and neural networks to the forensic area. This goal of this project is to build an efficient face detection method for videos starting by a database of famous actors' faces. In particular, in this project we aim at increasing the accuracy of the detection by combining the outputs of the different video frames. The project has been implemented in MATLAB.

1 Introduction

The face detection problem aims at building an algorithm capable of recognizing people's faces by extracting its features and comparing them to a database. The database must contain a large number of foreground photos where it is possible to clearly see the face and distinguish the various facial features. For this project, the provided dataset containing famous actors' faces has been used. Then, we need to build and train a neural network capable of recognizing the actors by comparing the test images to the features previously learned. Eventually, to face the video detection problem, I built a strategy able to combine the outputs of the detection in order to increase the final accuracy.

2 Dataset

As I mentioned before, the dataset contains a large number of foreground photos, possibly in high definition and with a good brightness-contrast ratio. Two functions, named *cropface* and *detectFaceParts* then use a Cascade Classifier to detect the faces and crop them in order to get a closer view on the features. The `vision.CascadeObjectDetector` is available in the Computer Vision Toolbox in MATLAB and it implements the Viola and Jones algorithm for the face detection, as it uses a sliding window (Haar features) across the face in order to detect the relevant facial features, such as the eyes, the nose and the mouth, as well as the face itself. Figure 1 shows an example.

Figure 1: Viola and Jones algorithm uses Haar features to detect relevant informations common to human faces, such as the fact that the eye region (black rectangle) is darker than the upper-cheeks (white rectangle).

Figure 2: Data augmentation applied to the face database.

To increase the accuracy, I also applied a data augmentation procedure to the images. This creates multiple cropped, rotated and scaled copies of the same image, as shown in Figure 2, so that the neural network can improve its training by relying on more samples.

Taking as an example the dataset available for Adam Sandler, starting from the 309 sample images of the original dataset, the algorithm was capable of extracting a total of 1772 images by means of the data augmentation, thus returning a more efficient input for the training. All the cropped images are rectangular RGB images (3 channels) of size around 188x188x3 pixels.

3 Neural Network

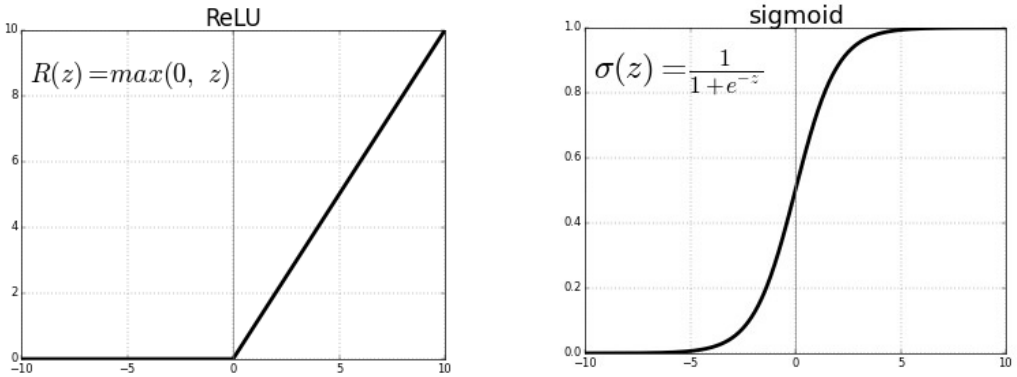
Neural networks are powerful computational models used for machine learning which are obtained by combining multiple perceptron models together. Convolutional neural networks (CNN) are a class of deep learning networks which include several convolutional hidden layers. Two models were tested in this project: a standard CNN and AlexNet, the latter being a convolutional neural network specifically designed for image recognition. All the input images from the database have been associated to a label which corresponds to the name of the actor in the picture.

For this project, the neural networks were tested to recognize a face out of 6 possible faces; trying to increase the number of possible outcomes for the predictions significantly increased the computational complexity and the machine was no longer able to give a result in a reasonable amount of time. Therefore, the actors used for this model are Adam Sandler, Alyssa Milano, Bruce Willis, Denise Richards, George Clooney and Gwyneh Paltrow.

3.1 CNN architecture

The CNN used for this project presents one input layer able to read as input a certain image, three 2-D convolutional hidden layers and a fully connected layer (output layer). The number of neurons which constitute the last layer is equal to the number of possible outcomes of the training.

Furthermore, the network uses the ReLU (Rectified Linear Unit) activation function (Figure 3a), which is, thanks to its efficiency, the most widely used one nowadays. In fact, differently from the sigmoid (Figure 3b), this function doesn't saturate at the value 1 (i.e. value 0 for its first derivative), thus avoiding most of the vanishing gradient descent problems usually associated with the sigmoid.



(a) ReLU

(b) Sigmoid

Figure 3: Plots of the ReLU and Sigmoid activation functions.

For the final evaluation and classification of the images, a Soft-Max function has been used. This function allows a multi-classification between the various outcomes of the network, giving as output the probability of every possible output of being correct (therefore the sum of the values will always be 1).

3.2 AlexNet Architecture

AlexNet instead presents eight layers: five convolutional layers and three fully-connected layers, as shown in Figure 4. As well as the standard CNN, it uses the ReLU activation function and it is able to perform multi-classification on the output values.

In addition, it makes use of a regularization tecnhique called *Dropout*, which consists in randomly turning off a certain number of neurons at every iteration in order to significantly decrease the probability of overfitting during the training.

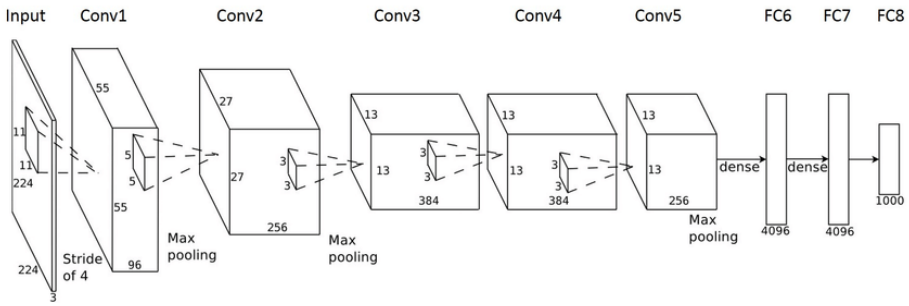


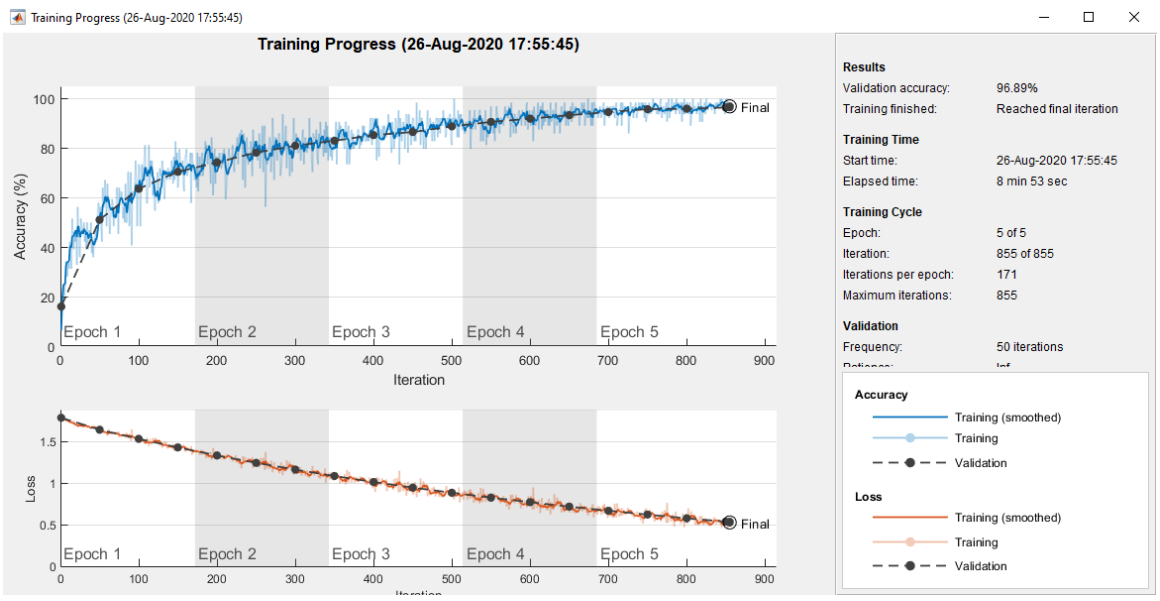
Figure 4: AlexNet architecture.

3.3 Training set up and Results

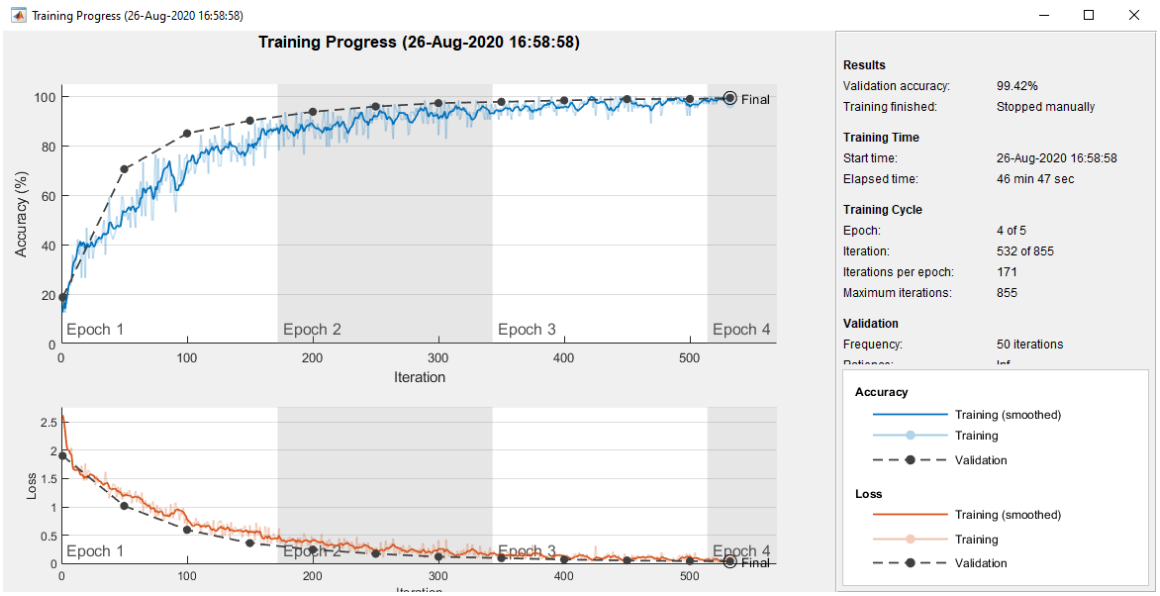
To perform the training, the dataset must be divided into training, validation and test sets. In this project, the training + validation sets constitute the 80% of the whole dataset, while the remaining 20% is the test set. The training is firstly executed over the validation set; then its performance is evaluated over the test set.

To obtain good accuracy and loss values, some hyperparameters need to be set accordingly: in particular, the choice of the number of epochs and the learning rate are very important factors since they determine whether the training will give successful results or not. For the CNN applied to 6 faces as possible outputs, it has been used a learning rate of 0.00003 over a total of 855 iterations.

For the AlexNet instead, the learning rate has been fixed to 0.000005 over 532 iterations. Further details are reported in Figure 5.



(a) Training of the CNN.



(b) Training of AlexNet.

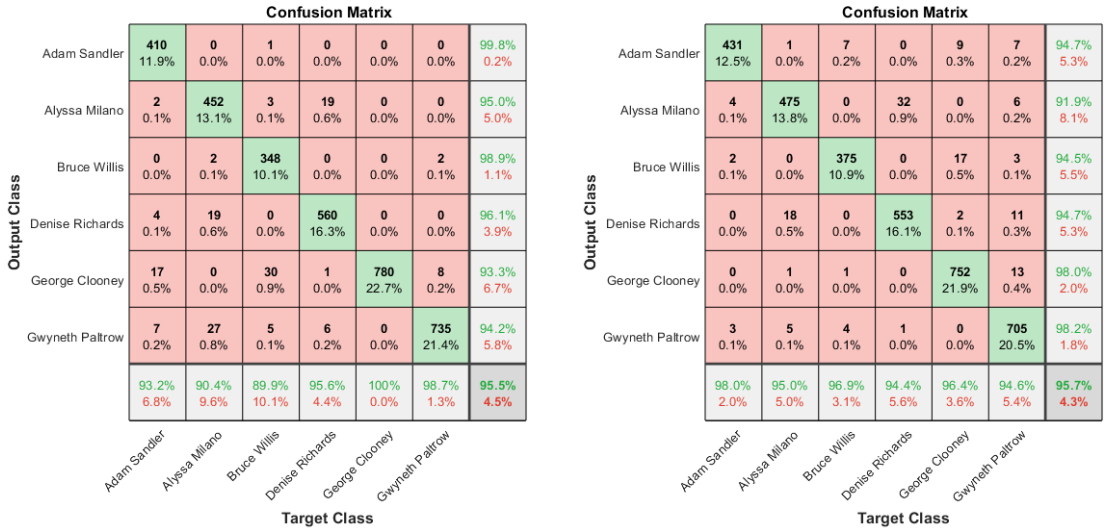
Figure 5: Training details (6 neurons in the output layer, that means 6 possible outcomes).

AlexNet, compared to the standard CNN, gives a much better output in terms of validation loss and accuracy, but requires a lot more computational power. Its training was also manually stopped in order to avoid overfitting of the data.

The results of the training over the test data are reported below in Figure 6. The confusion matrix represents on the rows the output classes obtained by the training and on the columns the target classes know a-priori. Each cell compares the two values and gives an estimate of the accuracy over the test set.

It is possible to notice that the overall accuracy is similar (around 95.6%): the AlexNet gives better results for the first three target labels, but the CNN prevails on the other three. In particular, for the Adam Sandler's target class, AlexNet solves much better many conflicting output values, especially with the George Clooney output.

Finally, the weights and biases of the networks are stored in a `.mat` file so that the results obtained with the training can be easily re-used for further investigations and tests. In this case, the files were used to test the training over the frames extracted from a video file.



(a) CNN

(b) AlexNet

Figure 6: Confusion matricies for standard CNN and AlexNet.

4 Face Detection in videos

In the following section I will discuss some of the possible applications of the face detection procedure over video files. These videos are considered as image sequences (i.e. frames): for every frame of the video the algorithm needs, in the first place, to detect the face. Then, it should classify it using the network previously trained and predict the output label. However, there are some problems that need to be dealt with: for example, during the video the subject could move is head thus making the classification inaccurate or, alternatively, other faces might be present in the frame hence yielding difficulties in finding the target face. For this purpose, the project needs to overcome these troubles and increase the final accuracy of the detection.

4.1 Tools and code structure

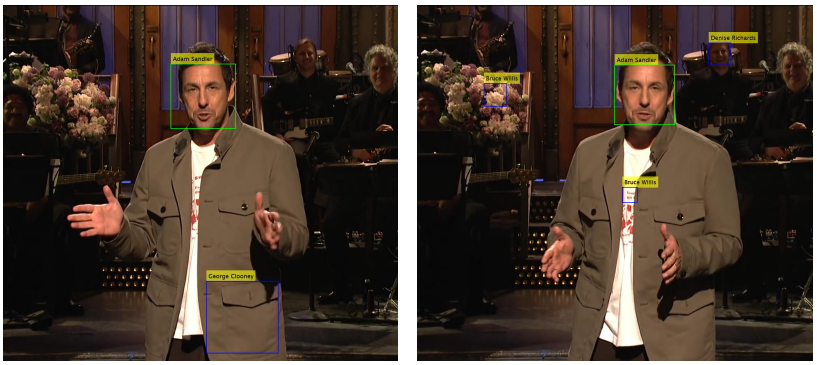
Figure 7: First frame from the sequence.

The video that has been taken into consideration for this project is from an Adam Sandler's monologue at Saturday Night Live show. Also, a small sequence of 9 seconds has been extracted in order to speed up the computation. A frame of the video is shown in Figure 7: notice that in the background there are many other faces which are hard to distinguish due to the poor illumination, while Adam Sandler in the foreground is clearly visible. To detect the faces, two functions already implemented for the lab experience have been used: *cropface* and *detectFaceParts*, already mentioned in Section 2. The latter turned out to be more efficient in the detection yet more computationally demanding. It also returns facial features such as eyes, nose and mouth locations. These two functions have been tested once at a time and will be compared again later in this report as they give different outputs.

Function *videoFaceDetection* uses one of the two face detection functions to detect the faces in the frames and select the best match with the target, thus using a classifier and returning the obtained predictions and scores. Finally, *faceDetection* runs *videoFaceDetection* and computes the accuracy of the detection.

4.2 Face extraction

For each frame of the sequence, one of the two detection functions are applied. *cropface* detects just Adam Sandler's face most of the times, but for some frames it wrongly identifies the left pocket of the jacket as a face instead (false positive, see Figure 8a). On the contrary, *detectFaceParts* detects also the faces in the background, but more frequently it identifies wrong objects as faces (e.g. the flowers in the background or some parts of the T-shirt, as shown in Figure 8b). Also, in some frames both functions do not detect any face at all (false negatives); this occurs most of the times when Adam Sandler's face is not clearly visible (e.g. he is not looking straight to the camera and his head is pointing downwards).



(a) *cropface* (b) *detectFaceParts*

Figure 8: Some of the errors found on the detection functions.

4.3 Face classification

Once the faces have been extracted from the various frames, it is necessary to classify them with respect to the network training that was previously discussed. In order to do this, the function *classify* of the MATLAB Deep Learning Toolbox has been used. This function returns two values: the **prediction** and the **score**: the former is the predicted label, while the latter is an array containing the probabilities for every possible label of being the correct one (SoftMax function, the values sum up to 1). With this tool, it is possible to classify the faces and show the result on screen. However, since there is some loss in the test set for both the networks, the classifier eventually gives wrong results. In Figure 9a I show an example of misclassification: notice that the classifier predicts the wrong label (George Clooney instead of Adam Sandler). Sometimes, instead, it detects two Adam Sandler's faces (Figure 9b).

(a) Wrong label (b) Two identical labels.

Figure 9: Some of the errors found on the classify function.

In order to overcome these errors, the program, after having detected all the faces, selects the best match by looking at the scores. In this way it is possible to avoid some of the conflicts in the classification by selecting the face that gets the best match (see Figure 8a).

4.4 Results

To determine the final accuracy, the ratio between the correctly classified labels and the total number of frames is computed. In Table 1 the accuracies obtained for 6 possible output labels are reported. Notice that all the results are pretty accurate except for the one obtained by using *cropface* and the standard CNN. In that case in fact, the algorithm classifies almost half of the times Adam Sandler's face as George Clooney's. This problem could be fixable by performing a re-training of the network with a different learning rate, yet paying attention to not overfit, but for the purposes of this project I will discuss in the next section how to fix it by operating on the single frames.

In addition, it is important to highlight some notes regarding the computational complexity. The algorithm, using *detectFaceParts* and classifying with a standard CNN, takes 5 minutes to return the prediction and scores over a 9 seconds long image sequence (frame rate $\approx 30 \Rightarrow 271$ frames). *cropface* instead takes about 60 seconds. For what concerns AlexNet, finally, it takes $\approx$ 5,51 minutes for *detectFaceParts* and around 90 seconds for *cropface* to detect and return all the predictions. Therefore, it would be convenient to use AlexNet and apply the *cropface* function to have good results in a reasonable amount of time.

	CNN	AlexNet
cropface	54.41	91.05
detectFaceParts	89.33	94.02

Table 1: Accuracies (in %).

5 Optimization

In the following section I will discuss a possible strategy to increase the accuracy results. This strategy combines the outputs obtained by the various frames in order to reduce the number of misclassifications basing its computation on the correlation between the images, since every frame of the video is similar to the others (mostly similar to the adjacent ones). Furthermore, this method aims at solving all the possible conflicts that decrease the accuracy, such as the ones I showed previously, especially for the *cropface* classified sequences employing the standard CNN.

The idea is to detect the area in the frames where the target face is most likely to appear. Given the sample video used for this project (Figure 7), it is possible to notice that Adam Sandler's face doesn't appear in all parts of the image, but it covers mostly a small area in it. So, in this case, the classification of his face should be continous for the duration of the whole video - i.e. it is not possible that for some random frames George Clooney's face is detected around the same area.

To do this, I created a matrix of `weights` so that the algorithm can learn where the target face is located in the image and classify the right face as a consequence. Initially, the matrix, which has shape 1920x1080 (the size of the original frame, which means that every value is associated to a certain pixel of the image), contains values all set to 1. For every iteration of the algorithm, it then updates its weights around the area where the face is detected in that iteration by multiplying these values by a certain `rate`. In this way, since the algorithm knows in which area of the image the face is more likely to be detected, it is possible to update the scores of the classification and, hopefully, solve the conflicts between multiple labels which have similar scores. In fact, in the next iteration, the scores and, consequently, the predictions will be computed accordingly to the `weights` matrix.

Figure 10 shows an example: here I show the first frame of the video and the resulting output for the `weights` matrix, with the updating rate set to 1.01. Values reported in green are the values that have been updated since they belong to the area where the face was detected.

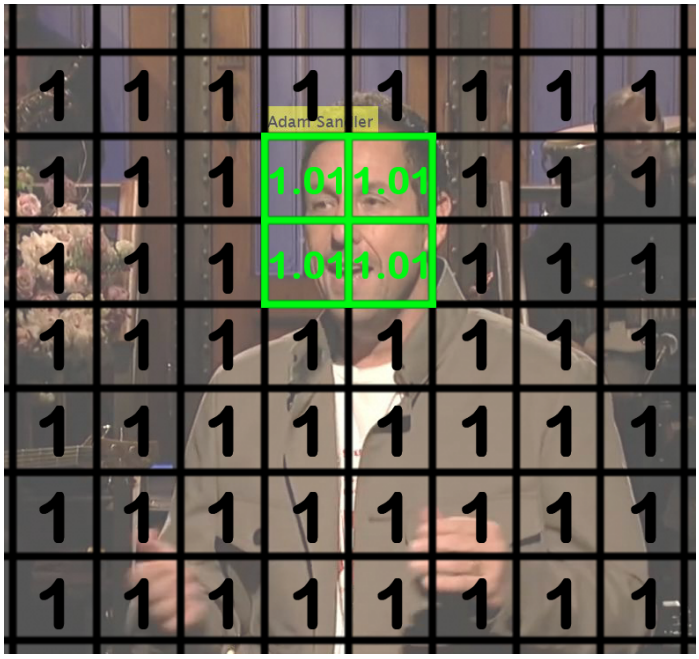


Figure 10: First iteration of the algorithm.

The matrix will then be the input for the next iteration of the algorithm (i.e. the next frame of the video, in which Adam Sandler's face is almost in the same position).

To update the classification scores accordingly, the sum of the `weights` values which fall in the area where the new face is detected is computed. Then, the average value calculated by dividing the sum for the size of the area. Equation (1) summarizes this process (W is the `weights` matrix and A is the area where the new face was detected). Finally, the update of the scores is carried out by multiplying the average by the initial scores of the classification.

The process is then iterated until the end of the video, thus allowing the algorithm to learn where Adam Sandler's face is most likely to be located in the video.

$$\frac{\sum Wwhere(W \cap A)}{size(A)} \tag{1}$$

Let's now take into consideration as an example the frame in Figure 9a. Table 2 reports the scores obtained with a normal classification (without matrix of weights) in the first row. It is possible to notice that the score for George Clooney is higher than Adam Sandler's but still pretty similar (≈ 0.27 vs. 0.29); in fact, adjacent frames have slightly bigger values for Adam Sandler.

In the second row instead I report the scores computed using the strategy explained above. Notice that the conflict is solved and Sandler's face gives as output the highest score. Of course, this is easier to see if we consider one of the last frames of the sequence, as, at that point, the algorithm has learned better were the correct face is located.

The algorithm also helps in the case of Figure 9b, since it recognizes the right Adam Sandler by means of its position in the frame.

	Sandler	Milano	Willis	Richards	**Clooney**	Paltrow
without weights	**0.2686**	0.0896	0.1869	0.0788	**0.2901**	0.0857
with weights	**0.3878**	0.0896	0.1869	0.0788	**0.2901**	0.0857

Table 2: Scores

Finally, Figure 11 shows the heatmap of the areas where Adam Sandler's face has been detected in the sequence. Warmer colors indicate areas with a more frequent presence of his face.

Figure 11: Heatmap of the face detection.

5.1 Results

Table 1 is now updated with the new results obtained by using this method with updating `rate` set to 1.05 (see Table 3). Notice that the algorithm is very effective when *cropface* is used (especially in the case of the CNN). This is due to the fact that errors that overcome using *cropface* are more frequent are more easily fixed with the application of this procedure.

	CNN	AlexNet
cropface	54.41 $\Rightarrow$ 96.69	91.05 $\Rightarrow$ 98.89
detectFaceParts	89.33 $\Rightarrow$ 94.42	94.02 $\Rightarrow$ 95.22

Table 3: New accuracies (in %).

The main issue in this image sequence occured for the frames where the actor has his head pointing downwards (Figure 12). In this case, no faces are detected thus leading to a loss in the final accuracy. This happens with both the detection functions.

Figure 12: Example of issue occured during the execution of the algorithm (no face detected).

5.2 Observations

To get good results, it is necessary to set the updating `rate` properly. In particular, it needs to be chosen accordingly with the duration of the video (i.e. number of frames), similarly to what happens when setting up the epochs and the learning rate of a neural network training.

In fact, a small rate would not have much influence in improving the accuracy, since it would not solve most of the uncertainties (underfitting). On the other hand, a high rate would detect always the same face around a certain area of the image (overfitting): suppose that George Clooney appears on stage during the video and positions himself in the area previously occupied by Adam Sandler. In that case the algorithm would still classify him as Adam Sandler, since the scores in that area would have increased too much. The same issue would occur also if the director changed the shot angle.

Table 4 reports the accuracy results obtained using different update `rates`, with the standard CNN being the reference network. Rate = 1 means that the algorithm has been applied without updating the weights.

rate	cropface	detectFaceParts
1	54.41	89.33
1.01	87,13	93.38
1.05	96.69	98.89

Table 4: Accuracies (in %) with respect to the different rates.

5.3 Further optimizations

In this part I will briefly discuss some ideas to further increase the accuracy. Not all of these techniques have been implemented in this project but they could be an interesting research topic for ulterior investigations.

5.3.1 Expansion of the weights region

Instead of just updating the weights in the area in which the face is detected, it could be possible to also update the weights matrix around that area, with values that become bigger as we get closer to the detected face. This would lead to a sort of frame prediction, thus giving more accurate results when the subject of the video moves in a certain direction. However, the accuracy would drop in case of the presence of multiple faces around the target one. In this case, we would not have much margin of expansion.

5.3.2 Saturation value

It would be appropriate to set a saturation (maximum) value to the weights matrix. During the videos, especially the longer ones, the values keep increasing at every iteration and might become too high, leading to wrong classifications. A saturation value would overcome this problem.

5.3.3 Multiple classification

We might also want to detect more than one target face in a video. To do this, it would be sufficient to add a third dimension to the weights matrix representing the weights (i.e. heatmap) associated to a certain target label. In the case of this project, the matrix would have shape 1920x1080x6 (where 1920x1080 is the size of the frames and 6 is the number of possible detected faces).

5.3.4 Intra-frame prediction

A slight modification to the method presented in this report could concern updating the weights matrix basing the prediction on the different between adjacent frames. As the example in Figure 13 shows, the two frames are very similar and the difference between the two matricies shows that most of the pixels have the same intensity level for both the images. Therefore, the weights matrix could be updated accordingly to this fact through a linear relationship.

To subtract an image from the other, the frames must be converted to grayscale images first. The output image of the operations should be mostly black, with some gray pixels in the areas where the difference between the frames is more evident. The weights are then updated accordingly to the number of non-black pixels over the total number of pixels of the cropped image.

This strategy works best when dealing with high frame rate videos. On the other hand, errors could occur if the subject moves his head quickly or the background changes.

Figure 13: Intra-frame prediction.

5.3.5 Motion estimation

It could be possible to predict where the subject is moving by applying a motion estimation between two adjacent frames. The Computer Vision Toolbox available in MATLAB provides an useful tool to do this (`vision.BlockMatcher`). In this way the performances of the algorithm could be improved by first looking for the face in the predicted area. If the face in the next frame falls in that region and the prediction returns the same face as the previous one, it would back up the hypotesis that the classifiction is correct.

Figure 14: Example of motion prediction.

5.3.6 False negatives correction

As I showed in Figure 12, in some frames no faces are detected at all. Considering three adjacent frames, if the first and the third detect the same face in a similar position, while the second one does not detect any faces, it could be possible to fill the gap by making a guess on the position of the face in the second frame (e.g. select an area between the other two). This could also be extended in cases where more than one frame is missing, but it would be appropriate to choose a maximum number of missing frames; otherwise, the algorithm would not be able to distinguish between a case in which some detections are missing from a case in which simply the shot angle is different and maybe there are actually no faces in the frame.

6 Conclusions

Face detection represents a very interesting and challenging research topic. The biggest challenge regarded the computational complexity of the algorithm: right now it would be too slow to perform live acquisition and detection, in the sense that the resulting video containing the results should be watched after having ran the algorithm.

During this project I thought of many strategies to use and I decided to concentrate my efforts on the one that in my opinion is the most appropriate to the final goal. However, all these methods can be combined together in order to obtain a flexible algorithm capable of adapting its classification to various different situations and problems (camera angle change, low frame rate, face not detected, etc.).

Nevertheless, this project shows that, when dealing with videos, face detection opens to a whole new multitude of opportunities to improve the already good performances obtained by neural networks.