University of São Paulo - ICMC
Bacharelado em Ciências da Computação
SCC0251 - Image Processing
Moacir Ponti

Partial Report - Facial Recognition

Julia Diniz — Nº USP: 9364865
Giovanna Oliveira Guimarães — Nº USP: 9293693
Lucas Alexandre Soares — Nº USP: 9293265

São Carlos - SP

June 30, 2017

Contents

1 Main Objective

The main objective of this Project is to be able to recognize multiple faces at different angles and positions and then perform functions such as:

- Change color and/or replace the key features (such as mouth, eyes and nose) by stylized components and objects
- enhance the images with the use of techniques such as sharpening, gamma and temperature filters.
- Apply distortion in an area
- Remove red eyes.

2 Input Description

The input of our Project consists of two parameters: a pre-trained shape predictor and an image of your choice. For testing, we used some random images from the internet. Any image can be used by passing the *im* parameter in the command line below:

python facial_landmarks.py - -shape-predictor *sp* - -image *im*

The same goes for the shape predictor (sp). The one used to produce our first results was dlib's default predictor, which was trained with the 68 point (coordinates) iBUG 300-W[1] dataset.

3 Steps Description

As previously mentioned, to obtain the preliminary results, dlib[2] library functionalities were used with their standard pre-trained shape predictor, included in the dlib's repository. The facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face. The indexes of the 68 coordinates can be visualized on the image below:

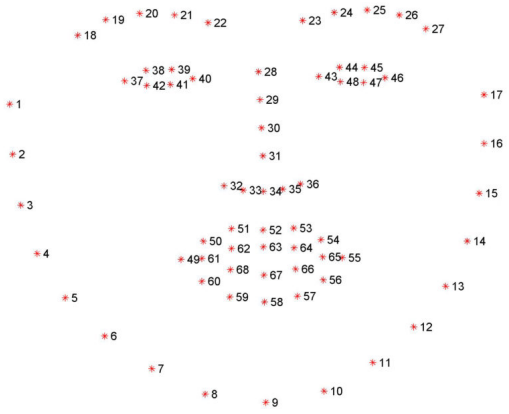
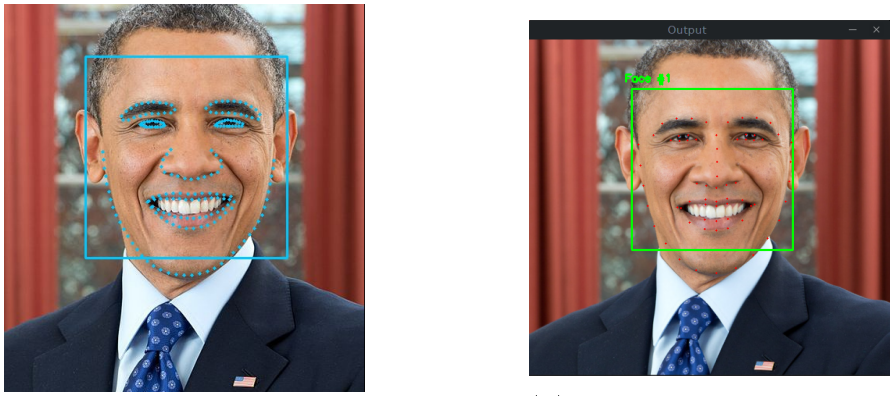


Figure 1: Visualizing the 68 facial landmark coordinates from the iBUG 300-W dataset

However, for the Project final phase, the predictor will be trained with images coming from the Helen[3] dataset with a 194 points model, which we expect better results.



(a) 194 landmark coordinates

(b) 68 landmark coordinates

Figure 2: Comparision between the 194(a) and 68(b) coordinates model

To train the predictor with the Helen dataset, first we got the annotations file included in the dataset and reorganized it in a xml format. Then new labels were added to the landmarks around the eyes, mouth, nose and eyebrows. The landmarks and labels were adjusted with dlib's imglab tool, included with dlib. Finally the predictor may be be trained with those 194 points and five labels. The training command is as follows:

python train_shape_predictor.py - -training-file *file* - -test-file *file*

This will generate a predictor.dat file which is the trained predictor data to be used with facial recognition. Now, the user may feed the predictor to the application with an image and

the program should return a bounding box in each identified objects (face, nose, mouth, eyes, eyebrow). In order to remove the red eyes, all the pixels of the eyes boxes will be traversed in order to check the intensity of their red tones. If the red channel value is much higher when compared to the blue and green channels, then we probably have a red eye there. If this is detected, we just normalize the red value to the blue and green levels, and thus remove the red eyes.
To apply the temperature filter, we will just rise the intensity of the red tones of an image if the user chooses to make the image hotter, or blue tones to make it colder.
Finally, the other filters (gamma and sharpening) will be applied the same way they used in classes assignments but in colored images. To do this, we will converet from RGB space to HSV and apply the filter in the Value channel since they only need to work with contrast, and not color directly.
The key feature replacement will be just a replacement of the identified object by one of the defaults and some smooth filtering/equalisation shall be applied to smooth out the contours.

4 First Results

Using the pre-trained predictor, we were already able to get a raster around each face identified in the input image and we can manipulate them already. The program can handle RGB and grayscale images, rotated faces and multiple faces in the same image.
The images below show exemples of our first results:

Figure 3: Example of a facial recognition performed in a colorful image

Figure 4: Example of a multiple facial recognition performed in a grayscale image

1 2 3

[1]Figure 1: http://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/
[2]Figures 3&5: https://www.facebook.com/apricotjaamu/
[3]Figures 4: https://br.pinterest.com/amylamps/multiple-images-photography-project/?lp=true

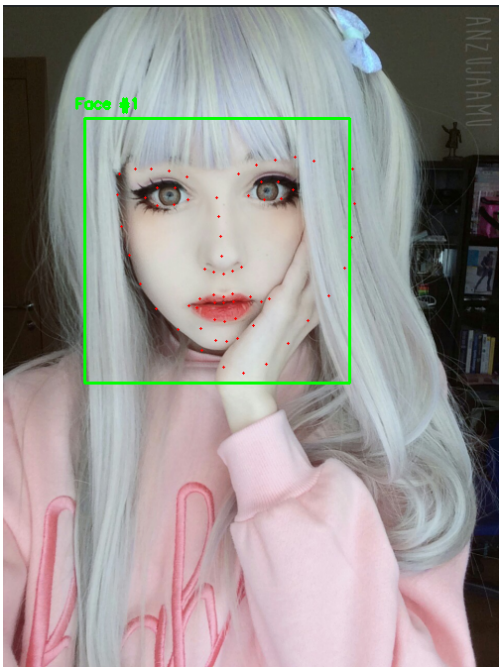


Figure 5: Example of a failed recognition that we expect to fix with the 194 point model

5 Final Results

As we didn't manage to construct the database and train the new 194 landmarks model in time, we decided to use cascade classifiers included with OpenCV to find some characteristics, like eyes. This is a lot more limited than what we previously wanted, however it sufficient for testing a lot of the features we envisioned.

5.1 Filters

Figure 6: Anzu3.jpg after applying Sharpen → Detail → Autocontrast → Equalize filters

5.2 Distortion

For applying distortion, first we transform the image to a grid and then to a mesh-like structure. This facilitates when moving and interpolating points in images. The user may define the grid resolution in X and Y axis separately, and also define the max distortion value, since it will be randomly applied in the image.

It is possible to obtain interesting effets by tuning the distortion parameters, like a glass or watery effect as seen in 7 and 8

Figure 7: Anzu3.jpg with distortion (x = 200, y = 200, distortion = 50)

Figure 8: Anzu3.jpg with distortion (x = 65, y = 35, distortion = 15)

5.3 Red Eyes

For red eyes removel, we used the Haar Cascade Classifier included in OpenCV to locate the eyes. Having the eyes, the program separates the red channel from the green and blue, apply a threshold in the red channel for a minimum red intensity that the user define as possible red eye, apply a closing operation in the image to remove noise and small "holes" and finally replace the red pixels with the mean of the green and blue channels. This way, the color seems to blend better with the eye iris. The green and blue channels are used as masks when replacing the pixels, to guarantee a smooth eye.

Figure 9: Original photo

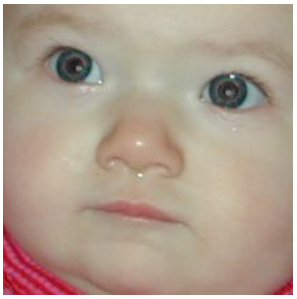


Figure 10: After removing redeye (threshold = 80)

6 References

iBUG: https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/
Dlib: http://dlib.net/imaging.html
Helen: http://www.ifp.illinois.edu/~vuongle2/helen/
Pillow: https://python-pillow.org/
Cascade classifiers: https://en.wikipedia.org/wiki/Cascading_classifiers
OpenCV: http://opencv.org/
Haar Cascades OpenCV: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.htm
Red eye removal: http://www.learnopencv.com/automatic-red-eye-remover-using-opencv-cpp-p