Face recognition

Axel Beauvisage, Carole Belloni, Adrian Kostkowski, Gareth Hughes, Domonkos Huszar

***Abstract*—**

I. INTRODUCTION

THE human face is one of the most popular characteristic which can be used in the biometric security system to identify or verify a user. Face is an acceptable biometric modality because it can be captured from a distance, even without physical contact of the user being identified. Thus the identification or verification does not require cooperation of the user. Recognition systems based on human face are used for a wide variety of applications, due to these benefits.[1] The recognition of the faces in different contexts enables various applications such as surveillance monitoring, social networking, and human-computer interaction.[2]

For example, in the recent event of Boston marathon bombings, images of the suspects taken from street surveillance cameras aided the FBI to identify the suspects. ¡¡¡06905824¿

Face recognition includes the studies of automatically identifying or verifying a person from an image or video sequences. Face identification refers to determining the ID of the person, given as a probe, from a large pool of candidates in the gallery. Face verification or face authentication is the problem of deciding whether a given pair of images are of the same person or not[2].

II. METHODS

A. Normalization and Haar feature-based cascade classifier

Preprocessing of the image is needed in order to have a good recognition. We'll use for this purpose equalized gray images to a better detection of faces during the normalization. We use a Haar features-based cascade to detect the face. The output will be the difference between the sum of the intensities on the whole picture and the intensities on the black part of the feature. The feature are scaled and move

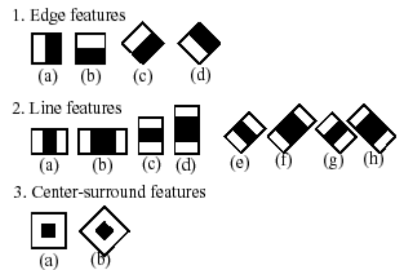


Fig. 1. Haar features used in openCV (including 45 features)

across the initial image. These are quickly calculated by using the corresponding intensity image. We use indeed two intensity images : one adding

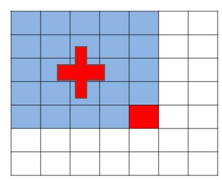


Fig. 2. Integral image : sum of the intensities of the value of left and up pixels of the current pixel

the left and up pixels of the analyzed pixel and an other intensity image adding the pixels in the corner of the current pixel made by two 45 lines. The cascade is then trained. Adaboost is used for this purpose on positive images (a face, an eye, a nose, a mouth). The result is a sequences of criteria of recognition following a cascade checking of the object. However, Haar will only recognize not rotated face. If we do not recognize a face, the face will be rotated and the same Haar process will be used. Once a face is detected we'll use again Haar features to detect the eyes or the mouth and nose. The angle remaining to have the eyes or nose and mouth aligned is then used to have these features aligned. The face is then cropped to the dimensions of the face detected by Haar features and resize to a specified size, so that every picture has the same

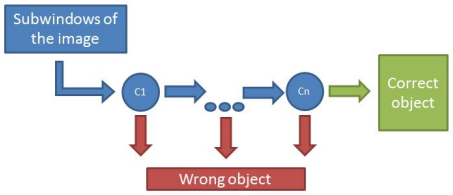


Fig. 3. Cascade

size.

B. Eigenfaces method

The Eigenfaces method is based on a principal components analysis from a training set of images. To get those, each normalized image is translated into a vector. The principal components will be then determined : They have the highest eigenvalue in the diagonalized correlation matrix (made out of the image and the mean values of the training set). After selecting the principal components (first eigenvectors called eigenfaces in this case), each image can be then be approximated with those weighed eigenfaces :

$$x \approx \hat{x} = \bar{x} + E_x * W, \text{ where} \tag{1}$$

x = vector representing the image,
$\hat{x}$ = approximated vector,
$\bar{x}$ = mean values of the vector during training,
E_x = Matrix of eigenfaces (principal components),
W = Matrix of weights of eigenfaces

W is particular to each image and is calculated to minimize the distance between x and $\bar{x}$:

$$W = E_x^T(x - \bar{x}) \tag{2}$$

The face recognized will be then the one in the training having its weight matrix W the closest to the weight matrix of the current face analyzed.

C. Fisherfaces method

The Fisherfaces method is an improved version of the Eigenfaces method that uses Linear Discriminant Analysis (LDA) rather than Principle Component Analysis (PCA) to find features that maximise the variance in the data. The problem with PCA is that it can consider variances in each picture, such as changing light sources as principle components and therefore not actually contain discriminating information about each face. To prevent this, Fisher's LDA method maximises the ratio of between-class scatter

$$\sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{3}$$

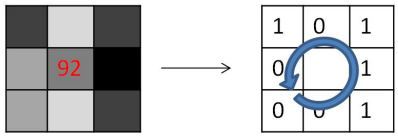
and within-class scatter.

$$\sum_{i=1}^{c} \sum_{x_j \epsilon X_i} (x_j - \mu_i)(x_j - \mu_i)^T \tag{4}$$

Similar classes(faces) should therefore group together and dissimilar classes should be far away from each other. Fisherfaces should produce lower errors than the Eigenfaces method and also be better at recognising faces in varying lighting conditions and different facial expressions.

D. Local Binary Pattern Histogram

The local binary pattern histogram metandhod uses the local feature of the face. First, the image is sliced into squares. To avoid problems of scale, The neighbours of a pixel analysed will be the ones on a circle with a varying radius and this pixel as center. The intensity of the central pixel is then compared to the ones of its neighbours. (higher values become 1, lower ones become 0). We then compute the new value of the central pixel, being the sum of powers of two combined with the value of neighbor taken clockwise. The point is to spot particular patterns of neighbor such as edges, lines, corner, flat areas.



0x1+0x2+1x4+1x8+1x16+0*32+1*64+0*128
=92

Fig. 4. Analysing the local structure

The histograms are then added (and not merged) for each part of the image. The resulting vector is then compared to the vectors obtained during the training. A fixed number of the closest vectors to the resulting vector is chosen and the person assigned to the greater number of these vector will be associated with the initial image.

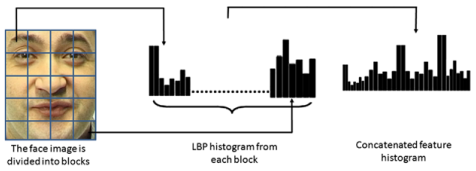


Fig. 5. Adding histograms (source : http://what-when-how.com/face-recognition/)

E. RBF and Convolutional NeuralNetwork

NeuralNetwork stuff here

III. Implementation

A. The Graphical User Interface

To make our program simple and easy to use, we have created a Graphical User Interface(GUI). To do so we have used the Qt library and because it can generate problems on some computers, the program can also be run without GUI by specifying the training and testing folders in the command line. The GUI is divided into 3 main parts: the main window, the result dialogBox and the ImageCapture program.

1) The main window: It is the first window to appear when the program is executed. It contains the following functionalities:

- A menu to load and save classifiers in a file (the classifier selected in the list of methods will be used) and to run the ImageCapture program.
- A path selection tool to specify where is are the training and validation folders. The paths can be modified directly from the text inputs or by browsing it using the "..." button.
- A widget displaying the image retrieved from the webcam. A Rectangle has been added to help the user to center his face on the picture.
- An output console embedded in the window.
- A normalization checkbox to apply the normalization on the images or not
- A progressBar to make sure that the program is still running while loading the images (unfortunately it was not possible to do the same for the training).

2) The result dialogBox: When a new picture is taken by the user, a new result window is created. This dialogBox call the predict function from the selected classifier and display the results to the user(image, predicted label and confidence level).

3) The ImageCapture program: - Drag'n Drop square - Big cross for alignment - Space Bar to record

4) The Webcam object:

B. Dataset

1) Databases used: During the project, we used different databases :

- The database provided by the AT&T laboratory in Cambridge
- The Yale database. These two first databases were mainly used to carry out the first tests as the images were already prepared (grayscale, right size, faces centered with different illuminations).
- The caltech database to test the preparation of the images and the methods on faces in front of different backgrounds, illuminations without any preprocessing.
- Databases on the faces of the people in the group project. (Personal database)

2) Acquisition of our own Dataset: Dataset used for training :

- 3 persons, 100 pictures for each from 1 take.
- 3 persons, 100 pictures for each from 3 take.
- 3 persons, 10 pictures for each from 3 take.
- 5 persons, 10 pictures for each from 5 take.

We firstly tried to do a little database with only three people selected from the group. We wanted to have the faces exposed to different illuminations and in front of different background to avoid a recognition valid only under certain conditions. The process was to make different faces (smiling, opening the mouth...) in different places. A video was then taken while moving the face and the computer in a partially lighted room in order to get different illuminations.

We have broken down our initial dataset into two folders: a training folder and a testing one. Firstly, we respected a 80%, 20% repartition between the training and validation dataset. However, there was too many images for each person in the training dataset and we experienced overfitting. Indeed, if the training and validation test was the same,

results were good. If we took other images for the validation, the results could be less efficient than a random choice (<30% in the first database), or recognize a type of illumination instead of a person.

The most efficient training set was approximately 10 images taken in different conditions.

C. Preprocessing

It is known that Eigenfaces and Fisherfaces were sensible to rotation. To maximize the recognition, a preprocessing of the faces was made and consisted in getting a centered, straight head in a known size image. However the Haar detection can take up to 1 second, and the rotation and cropping on the initial colour image up to 0.3 seconds. (The colour image was kept in order to use the colour on other methods not yet implemented in the project). The time of processing prevent doing some real time recognition. The recognition with normalization of the images on a 3 person database jumped from 21% to 89% of recognized face.

To improve our results, we also try to equalize the histogram of the gray image before the analyze of it. As there was no improvement of recognition and as it took some processing time, we stopped the equalization.

D. Methods

E. Testing parameters

F. Loading and Saving

To have an easier loading of the model through the GUI interface, there is only one file to load. It is a merge of two files : An xml file representing the model itself and a csv file representing the linking between labels (integer) and names associated.

IV. Results

The result goes here.

V. Conclusion

Before making a conclusion I will first wait for you to finish this paper.

some ideas : Face recognition is a tough problem But we manage to do something

Appendix A
Important code parts

Appendix two text goes here.

References

[1] B. Jozer, F. Matej, O. Lubos, O. Milos, and P. Jarmila, "Face recognition under partial occlusion and noise," in *EUROCON, 2013 IEEE*, July 2013, pp. 2072–2079.

[2] M. Kafai, L. An, and B. Bhanu, "Reference face graph for face recognition," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 12, pp. 2132–2143, Dec 2014.

[3] A. A. Farag and S. Y. Elhabian, "A tutorial on data reduction linear discriminant analysis," 2008.