Face-recognition methods comparison

With OpenCV and Deep Neural Network.

G. F. Poli, mat.766577

University of Bari "Aldo Moro", Computer Science course, Computer Vision report

June, 2022

Abstract

This work is proposed as a comparison between the known methods for a face recognition system proposed by OpenCV. Here are implemented all three OpenCV face-recognition algorithms (EigenFaces, FisherFaces and LBPH) and also a Deep Neural Network in order to assest which one performs better on the same dataset. Haar cascade has been used for facial identification.

1. Introduction

Face recognition is an easy task for humans. Experiments have shown, that even one to three days old babies are able to distinguish between known faces. So, how hard could it be for a computer? Well, it's not that easy.
Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition and as we speak, OpenCV since version 2.4 comes with the FaceRecognizer class for face recognition.
The currently available algorithms are:

- Eigenfaces;
- Fisherfaces;
- Local Binary Patterns Histograms;

The first true step to facial recognition is Facial Detection. The face-detection is performed with the Haar Cascade classifier (first in the dataset and then in the test images). With the classifier the starting dataset (which included more than 100 celebrities and 10000 images) has been cleaned for semplicity reasons but also because the dataset included several false positives, so It has been narrowed down to about 2500 images for training and about 500 for validation divided almost equally in 22 classes .

2. Haarcascade

Haar Cascade classifiers are an effective way for object detection in particular for face detecion. This method was proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features".

2.1. How does it work?

Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

Hence initially, the algorithm train the classifier. Then we need to extract features from it and for this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs, even a 24x24 window results in over 160000 features). But among all these features we calculated, most of them are irrelevant. For example, consider the image below.

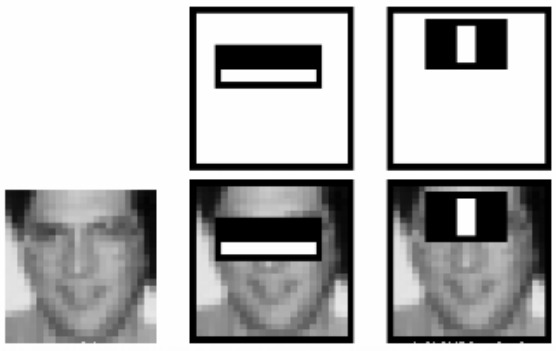

Figure 1: Haar features

The top row shows two good features:

- The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks;

- The second feature selected relies on the property that the eyes are darker than the bridge of the nose;

But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 60000+ features? It is achieved by Adaboost.

2.2. Adaboost

For this, each and every feature are applied on all the training images. Then the features with minimum error rate are selected, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated and also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. They're called weak because they alone can't classify the image, but together form a strong classifier.

In any way, in an image, most of it is a non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. This way, we spend more time checking possible face regions. For this it is introduced the concept of Cascade of Classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. If a window fails the first stage, then it is discarded, otherwise it goes through the next stage until the last one which tells us that in the image there is a face.

3. Methodology

As already said at the beginning, this work is a comparison between the OpenCV methods for face recognition and a DNN.

3.1. Set-up

In order to get the most reliable results it is necessary to have the same train dataset for every method which needs to be applied on the same test images. For this purpose the project goes through every folder and every file in the dataset, filtering the imges: It discards every picture in which does not find any face, and overrides the images in which there is at least a face with the roi of the face cropping out the background, in order to feed the algorithms with cleaner data.

Created the dataset, the methodology for the three OpenCV algorithms is the same both in training and in testing:

- Training:
  - It is created the respective recognizer;
  - The labels are created and it is assigned a label for every image starting from the folder's name, saving these informations in a .pickle file;
  - It is saved a resized grayscaled roi of the face (if found by the haarcascade);
  - In the roi are detected the eyes, which are used to allign the face overvriting previous roi;
  - The new roi is appended in the train array, as long as the labels in the labels array;
  - Then the train is started and the results are saved in a .yml file in order to be imported just once and not retrained at every usage;
- Testing:
  - Labels are imported;
  - For every image in the test folder it is taken a resized grayscaled roi of the face (if found by the haarcascade);
  - The roi is allinged detecting the eyes;
  - The prediction is made by the chosen recognizer (which returns a label and the confidence);
  - Depending on the confidence the recognizer returns the predicted label or "unknown";

Please note that confidence here is not to be inteded as accuracy, confidence measures the distance between the query image and the closest train sample, in addition to that, ditances are calculated differently among the algorithms: EigenFaces and Fisherfaces use the Euclidean distance, LBPH uses the chi-squared distance.

Now that how to detect a face has been explained as long as how it is intended to use the algorithms, let's talk about the three avaiable algorithms that OpenCV proposes.

4. Eigenfaces

The Eigenfaces method takes a unique approach to face recognition: A facial image is a point from a

high-dimensional image-space and a lower-dimensional representation can be found. So, The problem with the image representation we are given is its high dimensionality. Two-dimensional p×q grayscale images span a m=pq-dimensional vector space, so an image with 100×100 pixels lies in a 10,000-dimensional image space already.

The question is: Are all dimensions equally useful? The answer is no.

4.1. PCA

For this reason it is now introduced the Principal Component Analysis (PCA) proposed by Pearson and Hotelling to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

4.2. How Eigenfaces works

After computing, the mean, the covariance matrix and the eigenvectors (in which there are the ordered eigenvalues, that are the special set of scalar values that is associated with the set of linear equations of the matrix) the Eigenfaces method then performs face recognition by:

- Projecting all training samples into the PCA subspace;
- Projecting the query image into the PCA subspace;
- Finding the nearest neighbor between the projected training images and the projected query image;

Eigenvalues and eigenvectors allow to "reduce" a linear operation to separate, problems simplifying them. For example, if a stress is applied to a "plastic" solid, the deformation can be dissected into "principle directions"- those directions in which the deformation is greatest.

It can be seen, that the Eigenfaces do not only encode facial features, but also the illumination in the images (see the left light in Eigenface#4, right light in Eigenfaces#5):

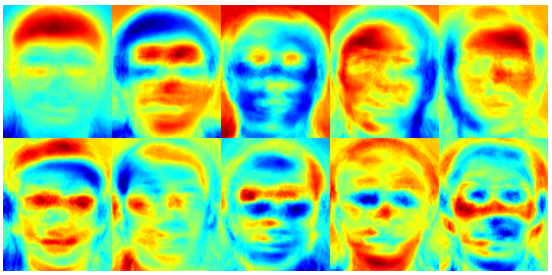

Figure 2: Eigenfaces light encoding

Taken into account the dataset created at the beginning of the project, the EigenFaces method does not perform so well as expected in fact it only guesses 6 people out of 22.

The same behaviour is observed with the FisherFaces algorithm, which for instance performs even worse, with only 3 correct guesses out of 22.

Since there's a similar reason to explain these results let's first analyze the FisherFaces method and then discuss the outputs of the two algorithms.

5. Fisherfaces

The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information may be lost when throwing components away. Imagine a situation where the variance in your data is generated by an external source, let it be the light. The components identified by a PCA do not necessarily contain any discriminative information at all, so the projected samples are smeared together and the classification becomes impossible.

5.1. LDA

For this reason the Linear Discriminant Analysis is introduced as it performs a class-specific dimensionality reduction. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation

5.2. How Fisherfaces works

The Fisherfaces method learns a class-specific transformation matrix and the Discriminant Analysis instead finds the facial features to discriminate between the persons. The Fisherfaces allows a reconstruction of the projected image. But since it has only identified the features to distinguish between subjects, you can't expect a nice reconstruction of the original image. The differences may be subtle for the human eyes, but it should possible to see some differences:

Figure 3: Fisherfaces reconstruction of faces

6. Mid-results

So, why did the implemented models get such bad results? The EigenFaces and FisherFaces methods need a dataset and a test set under specific conditions: the variance must be as lower as possible and at the same time it has to be also as meaningful as possible. Let's try to explain this visually taking a look at the AT&T database:

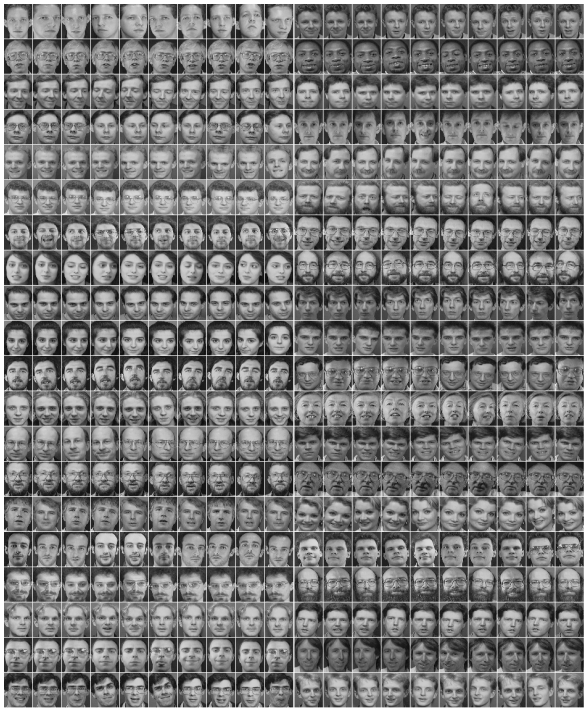

Figure 4: AT&T database of faces

In the 400 images the variance is very little, but still meaningful because all the pictures are taken at same height, with the same positions and the same light, which makes very easy to work with the two methods, because the variance will point out just the principal feature of every person.

Let's now take a look at some of the images in the dataset of this project:

Figure 5: Some of the images within the used database

As we can see, light, poses, expressions are completely diffrent from image to image even within the same class. This invalidates the work of the Eigenfaces and Fisherfaces algorithms, because they rely on the meaning of the pure variance between images which is achieved if the pictures are taken just frontally or with very few degree of rotation and with the same condition of positions and light. However, the dataset for this project is prone to be used as a benchmark for real time face recognition, in which all those conditions are rarely encountered. This does not mean that Eigenfaces and Fisherfaces are not useful at all, but since the results obteined it should be safe to say that these methods shift their focus on authentication more than on recognition.

Everityhing said until now turns out to be different taking into account the LBPH method.

7. Local Binary Pattern Hisograms

The local binary patterns histograms method is by far the most popular method for live and real-time face recognition.

7.1. How LBPH works

Real life isn't perfect. It's just not possible to guarantee perfect light settings in every image or 100 different images of a person. So some researchers concentrated on extracting local features from images. The idea is to not look at the whole image as a high-dimensional vector, but describe only local features of an object. The features you extract this way will have a low-dimensionality implicitly. But it is soon observed that the resulting image representation doesn't only suffer from illumination variations, but think of things like scale, translation or rotation in images - the local description has to be at least a bit robust against those things. The basic idea of Local Binary Patterns Histograms is to summarize the local structure in

an image by comparing each pixel with its neighborhood. Take a pixel as center and threshold its neighbors. If the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. You'll end up with a binary number for each pixel, just like this:

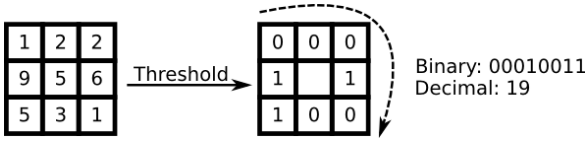

Figure 6: Tresholding neighbors with LBPH

This description enables you to capture very fine grained details in images. In fact the authors were able to compete with state of the art results for texture classification. It is now possibile to identify certain patterns on the image that were not recognized before by the other methods, such as:

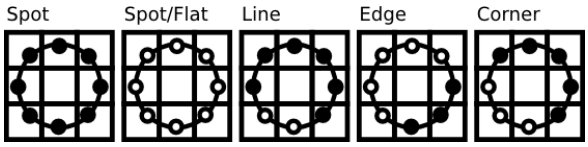

Figure 7: Possible identifiable patterns

By definition the LBP operator is robust against monotonic gray scale transformations. It is easily verified just by looking at this image:

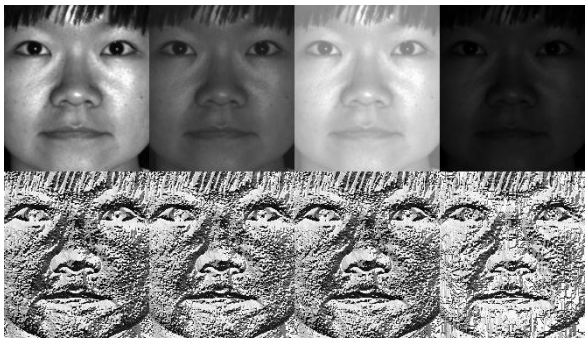

Figure 8: LBPH robustness against grayscale tranformations

As we can see most of them look basically the same. That's why this method works well also in this project, in fact it outperforms the previous methods guessing correctly 17 people out of 22.

8. DNN

In order to have a more complete insight on the face recognition solutions it has been also implemented a Deep Neural Network and it has been trained on the same dataset.

8.1. What is a DNN?

A DNN consists of a hierarchy of layers inhereby each layer transforms the input data in a more abstract representation (ie. edge, nose, face) and the output layers combine those representation into making predictions. The key advantage with respect to a simple CNN is that a DNN can automatically learn complex representations using vast amout of data. Hence, starting from a CNN in which neurons are locally connected, it applies specific convolutional kernels (filters) to a window (called receptive field) of the input image until high-level features are extracted. We can identify four main operations:

1. Convolution:
   The main building block of a CNN that applies multiple trainable filters and produces an activation map(feature) for every filter. Each filter has its own weight and it's shared between every neurons related to the filter. So we can learn only one set of weights that is shared among all neurons related to the same filter (weight sharing). The number of filters is called depth of the output volume.
2. Non-linearity (ReLu):
   This is needed after every convolutional operation to learn non-linear representation of data. The most used function is the ReLu (rectified Linear Unit). It is an element wise operation that replaces all negative pixels values in the feature by 0.
3. Pooling:
   Pooling reduces the dimensionality of each feature map retaining most of the important informations, resulting in a reduced resolution output feature map robust to small variations. So it makes the representation smaller and more manageable.
4. Classification:
   the classification of the input image into one of the classes which the CNN is trained on.

But before the classification it has been defined also three dense layers that use the features coming from the convolution and pooling for classifing the input image through a multi layer perceptron that uses both ReLu and Softmax function in the output layer.

8.2. Structure of the DNN

In the code it is possible to see that it has been implemented a DNN with 5 conv2D layers each one followed by a pooling layer. Then the output has been flattened to a 1D array and in between two dropout functions of 0.3 it has been implemented a 512(output size)-dense layer and another

128-dense layer. At the very end the last 22-dense layer. The model has been compiled with adam optimizer and a patience of 12 to prevent overfitting, and it has been trained on the usual database, plotting accuracies and losses. Let's talk about its results:

- On the training set it performed with 0.14 loss and 0.95 accuracy;
- On the validation set it performed with 1.34 loss and 0.68 accuracy;
- Which bring us to a 0.77 accuracy on the test set with 17 correct guesses out of 22;

Hence, it has been build a DNN that performs exactly as the most popular OpenCV method, LBPH.

9. Conclusions

Eigenfaces and Fisherfaces take a somewhat holistic approach to face recognition treating data as a vector somewhere in a high-dimensional image space. But we all know high-dimensionality is bad, so a lower-dimensional subspace is identified, where (probably) useful information is still preserved. The Eigenfaces approach maximizes the total scatter, which can lead to problems if the variance is generated by an external source.

In order to preserve some discriminative information we applied a Linear Discriminant Analysis as described in the Fisherfaces method which should work great at least for the constrained scenario we've assumed with the AT&T face dataset.

LBPH tries to encode a local structure in an image by comparing each pixel with its neighborhood, and using this encoding it is capable of identifying certain patterns, remaining still robust against various lighting conditions.

The DNN calculates its learnable parameters, which for instance at the very end are about two millions, simplifies the input through the pooling layers and the dropout function, implements three dense layers and once it's fully trained it's ready to make its prediction on the test set.

In any case, these are the results with a test set of 22 images:

Algorithm	Correct Guesses	Unknowns	Errors	Test-set accuracy
EigenFaces	6	1	15	0.27 %
FisherFaces	3	0	19	0.13 %
LBPH	17	0	5	0.77 %
DNN	17	0	5	0.77 %

At the end of this work it is clear that first of all you should have right in mind what are the goals you intend to achive with face recognition. It could be simple recognition, it could be authentication or any other application of it. Once the goals are fixed it's crucial to shape your dataset in order to feed the algorithm you choose with the most correct data it needs, then in order to improve the recognition performance, there are many things that can be done, some of them being also fairly easy to implement. For example, in this project could be added color processing or edge detection. Usually it's possible to improve the face recognition accuracy also by using more input images, by taking more photos of each person, particularly from different angles or lighting conditions. If it is not possible to take more pictures, there are several techniques that can be used to obtain more training images, by generating new images from your existing ones through data augmentation:

- You could create mirror copies of your facial images, so that you will have twice as many training images;
- You could translate or resize or rotate your facial images slightly to produce many alternative images for training, so that it will be less sensitive to exact conditions;
- You could add image noise to have more training images that improve the tolerance to noise;

Hence, it is important to remember to have a lot of variation of conditions for each person, so that the classifier will be able to recognize the person in different lighting conditions and positions, instead of looking for specific conditions. But its also important to make sure that a set of images for a person is not too varied, such as if you rotate some images by 90°: this would make the classifier to be too generic and this will cause bad results. It's also important to crop away as much of the background as you can, such as by only using a small section inside the face that includes as little as background as possible. For this reason it is essential a good pre-processing plan in order to feed the algorithm with a good and clean dataset.