Face Detection

Yu Du, Bincheng Huang, Siyi Liu

August 10th 2018

Abstract

Currently, face detection has become a highly developed technology, it is widely used in multiple areas, includes face recognition, camera focus, face locker, etc. The aim of the projects to explore face detection algorithm that can precisely detect and locate faces in image.

1 Introduction

In the project, two face detection algorithms are developed based on Local Binary Patterns Histogram(LBPH) and Haar. For LBPH-based algorithm, the first step is to extract the image pattern with LBPH algorithm. Then, two thresholds are set to calculate the probability of face in image pattern. After that, the slide window is used to detect faces in image. For Haar-based algorithm, Adaboost and cascade classifier algorithm are used to detect face.

2 Local Binary Patterns Histogram

The goal of face detection is to detect and locate faces in image, in order to extract human face to use in other areas. Nowadays, there are many different algorithms to accomplish face detection or recognition, such as Eigenfaces,Fisherfaces, Scale-invariant Feature Transform (SIFT), and Speed Up Robust Features (SURF). In this section, LBPH-based face detection algorithm is introduced.

LBPH algorithm is the combination of Local Binary Patterns (LBP) and Histograms of Oriented Gradients (HOG) descriptor. LBP is an easy but powerful way to extract and label the pixels of an image. Using the LBPH, we can easily represent a face images with just one simple vector.

2.1 Local Binary Patterns

Local Binary Patterns (LBP) was first introduced by Ojala et al and it is designed to be a texture analysis for gray-scale image. [1] [2] To detect faces in a RBG (colored) photo, we have to convert the image into gray-scale image at first. For each pixel P_{ij} , it is a vector that contains three values, which is to represent the degree of red, blue and green. We convert the RBG image into gray-scale image by:

$$G_{ij} = (0.2989, 0.5870, 0.1140)^T \bullet P_{ij}$$

Where G_{ij} represented the corresponding pixel in gray-scale image. The LBP operator is going to compare the center value with its P neighbor values on the circle with radius R and assigns neighbor value as 1 if center value is bigger than the neighbor, assigns 0 on the contrary. In this case, we set P = 8 and R = 1 which means that we consider a 3×3 check. The LBP operator labels the center pixel by thresholding the 8 neighbors. For each center pixel, the LBP operator outputs an 8 bits binary number and we convert it into a decimal between 0 and 255 as a result.

We utilize the following notation to describe the LBP operator:

$$LBP_{P,R}(x, y) = \sum_{i=1}^{8} sign(G_i - G_c)2^i$$

Where $LBP_{P,R}(x, y)$ is the results of the LBP operator, (x, y) is the coordinate of center pixel in the 3×3 check. Gc is the center value in gray-scale image, Gi is the neighbor value in gray-scale image.

The LBP operator reduce the influence of illumination and returns the texture by considering every pixel in an image exclude the boundary pixels. Figure 1 shows the demo of LBP operator for one pixel.

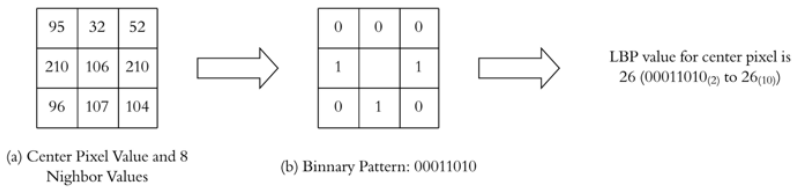
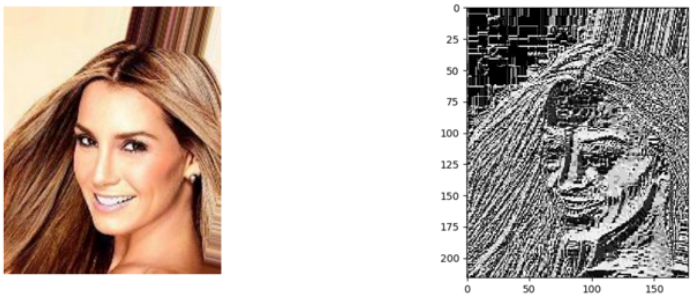


Figure 1: The Procedure of LBP for One Pixel

Figure 2 shows the LBP result of an example RBG image.



(a) Original RBG Image (b) LBP Result Image

Figure 2: LBP Result for RBG Image

2.2 Extracting Histograms with LBP

Using the LBP result, we can generate a histogram for this image and formed a data vector to describe the patterns of the original image. The Histogram is about the frequency of the occurrences of LBP result for each pixel. From the last part, after doing LBP operator, the value of each pixel is between 0 and 255. Thus, the histogram contain only 256 positions.

However, we are not considering the whole image directly. We divide the image into several image pieces. We calculate the values of 256 positions for each image piece. Given Nx and Ny, representing the number of cells in the horizontal direction and vertical direction respectively.

Then, we concatenate all histograms of all image pieces to form a bigger histogram. For instance, if we set $\mathrm{Nx} = 4$ and $\mathrm{Ny} = 4$, we will have $4 \times 4 \times 256 = 4,096$ positions in the total histogram. Finally, we can use this total histogram to represent the image by just one data vector.

3 Applying LBPH to Train the Model

Applying LBPH to our face dataset, we can combine all data vectors extracted from image dataset. With these patterns, we can train our model so that we can summarize the pattern of a human face.

3.1 Distance Threshold

Given 1,000 images as training dataset, each image is about a whole human face, including eyes, nose, mouth, ears, etc. Moreover, there is no other object in the picture. We assigned the LBPH operator to each training image and got its pattern. Then, we got 1,000 patterns from our training dataset.

Supposed those 1,000 images are in order. We computed the Euclidean distance between the former one and the later one, so that we got 1,000 – 1 = 999 distances data. We did the sample mean for these 999 distances. Because the Central Limit Theorem, the sample means subject to Gaussian distribution. Then, from that Gaussian distribution, we can easily to found out our distance threshold at mean plus 1.28 times standard deviation.

Finally, we set the distance threshold at 0.0023 level, so that we have 90% accuracy for our training dataset.

3.2 Probability Threshold

There are 100 more training dataset came in, each one is the same format as we used in section 2.1. For each one in those 100 images, we calculated the Euclidean distance between it and those 1,000 images in section 2.1.

Compared these 1,000 distances with the distance threshold 0.0023, we count the number of the distances which is less than the distance threshold 0.0023. For this additional training dataset, each image can get the probability which is computed by the number of distance less than distance threshold over the 1,000.

For these 100 probabilities, similar to section 2.1, we did the sample mean and extracted the mean from the sample mean distribution. We selected the mean of the distribution as our probability threshold, which came out with 79.8%.

4 Face Detect Process

Now, we got both the distance threshold and probability threshold so that we can basically find out the human face in a digital photo.

4.1 Detect Test Photo

Supposed that there is a new image, and that photo has the same size as our training photo, we found out the pattern of this image by LBPH at first.

Then, we got the pattern for this test image and computed the distances with our training dataset 1, just like what we done in Section 2.2. We count the number of the distances which is less than our distance threshold 0.0023, and computed the probability by that number over 1,000. Compared with the probability threshold, it will be assigned as "This is a face" if it less than probability threshold 79.8%, or it will be assigned as "This is NOT a face". In this case, we had 83% accuracy.

4.2 Detect Face in Image

Section 3.1 is about detecting the small face. In this section, we are going to find out a way to detect a face in a digital photo which has larger size than our training dataset. We are going to use a sliding window to scan the digital photo. For each sliding window, we extract this small piece of image and judge whether this small image is a face through the procedure in section 3.1.

The sliding window will narrow down, then go over the whole image again to detect the human face until the window size is less than pre-defined size.

However, the sliding window can find out several possible faces, we are going to find out the window with highest probability and label this window image as face.

5 Haar Cascades

Despite of our own developed algorithm, we also implemented Haar Cascades algorithm to analyze and compare the deficiencies between algorithms.

5.1 Extract Haar-Like Features

Haar-like features are digital image features extracted from a certain image that follow the below patterns. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector.

As Figure 3 shown, each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

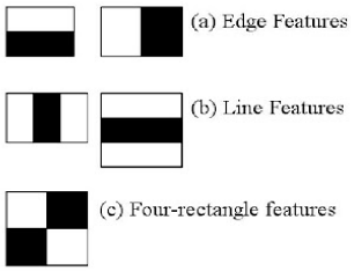


Figure 3: Haar-Like Features

For instance, if we are given a 24*24 pixels image, some of the features might look like Figure 4 shown:

Figure 4: Image with Haar Features

Where the first feature selected focuses on the property that the region of the eyes is often darker than the region of the nose and cheeks and the second feature selected relies on the property that the eyes are darker than the bridge of the nose.

5.2 Adaboost

For a 24*24 pixels, we can have 160,000 Haar features from all possible sizes and locations of each feature. And to train these features to form a model, we introduce a machine-learning algorithm called adaptive boosting.

What is Adaboost? Adaboost, short for Adaptive boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers.

For our face detection approach, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. 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. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found.

5.3 Cascades Classifiers

After we have finished the training process, we will result in a model that contains roughly 6,000 Haar features that can most distinguish if a 24*24 pixels given window is a human face or not.

And to detect if a given image has a human face, we implement cascades classifiers.

We take each 24*24 pixels window in a given image. 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, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

At last, we only need to combine all the face region and draw a rectangle around it, and our detected faces will be successfully captured.

6 Result

6.1 LBPH-based face detection

Currently, LBPH-based face detection algorithm has limited capability to handle complex environment such as multiple faces, half face or panting face. The Figure 5 shown a demo for our face detection algorithm. The result shown how we are approximately correct to add a frame for a human face. Also, several other test results can be found in code folder – test_result.

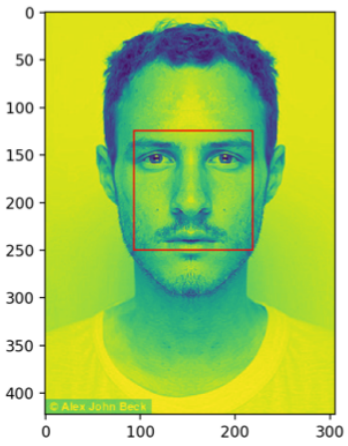


Figure 5: LBPH-based face detection demo

6.2 Haar-based face detection

We implement OpenCV package in python to accomplish the Haar Cascades face detection. Figure 6 is a demo for it. Compared with Figure 5, Figure 6 can perform better detection since the frame is including jar and ears. Moreover, Haar Cascades Algorithm has 93% - 95% accuracy based on OpenCV history record.

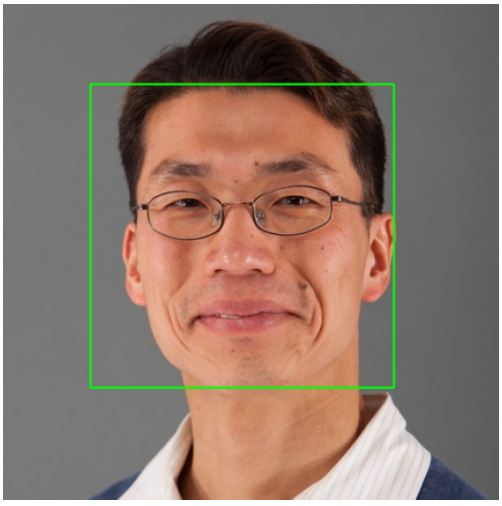


Figure 6: Haar-based face detection demo

References

[1] T. Ojala & M. Pietkainen & D. Harwood. *A comparative study of texture measures with classification based on featured distributions.* Pattern recognition, 1996.

[2] T. Ojala & M. Pietkainen & D. Harwood. *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.* IEEE Transactions on pattern analysis and machine intelligence, 2002.