FPT UNIVERSITY HCMC

Face recognition and tracking using LBPH algorithm

Computer Vision - SU24

Le Phan Manh Hung — SE182409

Le Long Song Thien — SE184516

Do Hoang Phuc — SE184737

1 Introduction

Facial recognition is a way of identifying or confirming an individual's identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time. Since its introduction, facial recognition has attracted significant attention and has been widely applied in many fields, including security, biometrics, and human-computer interaction. Over decades of technology development, many powerful deep learning algorithms have been born such as DeepFace, FaceNet, etc. However, in this project, we want to use classical-related algorithms to have the most general overview of face recognition.

Classical face recognition methods are divided into three types: global approaches (Global, such as Eigenfaces-PCA, Fisherfaces-LDA), approaches based on local features (Local Feature Based, such as LBP, Gabor Wavelets), and hybrid methods (Hybrid, which is a combination of two global approaches and an approach based on local features). **The Local Binary Pattern Histogram (LBPH)** method has ability to withstand noise and light variations, as well as its effectiveness in real-time face recognition.

In this project, besides recognizing faces using LBPH, our project also uses the **Haar cascade** algorithm - a classic algorithm that is not too complicated but effective - to detect human faces and **Kernel Correlation Filter** for tracking to decrease the number of computation and fps. Let's explore and discuss more in the following sections.

2 Problem Definition

2.1 Scope of the project

The scope of this project is to create a face recognition system based on the Haar Cascade and Local Binary Patterns Histogram (LBPH) algorithms as well as Kernel Correlation Filter. The purpose is to understand the advantages and disadvantages of these classical algorithms, thereby gaining a comprehensive overview of face recognition techniques in computer vision.

2.2 Definition of problem

While humans find face recognition easy, computers struggle with the process. Despite significant advancements in face recognition technology, several key challenges remain that need to be addressed to improve the accuracy and efficiency of these systems. Computers have to deal with numerous interfering factors related to treatment of images such as: different angles of view, lighting conditions, resolution differences, hair style, glasses, facial expression etc. To solve this problem, we use the LBPH algorithm for face recognition and Haar cascade algorithm - a classic algorithm that is not too complicated but effective - to detect human faces.

3 Method

Normally, the face recognition problem is divided into two main parts: **face detection** and **face recognition**. In this section, we will discuss the two algorithms used in this project.

3.1 Face detection with Haar cascade

Haar Cascade classifier is based on the Haar Wavelet technique to analyze pixels in the image into squares by function. This uses "integral image" concepts to compute the "features" detected. Haar Cascades use the Ada-boost learning algorithm which selects a small number of important features from a large set to give an efficient result of classifiers then use cascading techniques to detect faces in an image. Let's explore some Haar-Features.

3.1.1 Haar Features

Gathering the Haar features is the first stage. Haar features are nothing but a calculation that happens on adjacent regions at a certain location in a separate detecting window. 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.

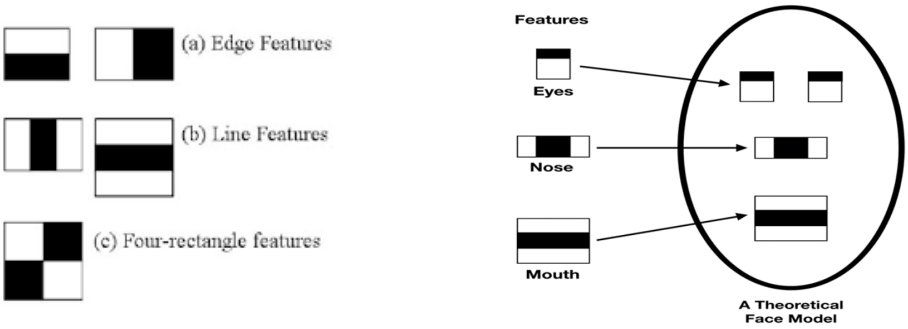


Figure 1: Haar Features.

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 over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image

3.1.2 Integral Image

Integral image is a concept that uses the cumulative sum of pixels above and to the left of the current pixel cell. Creating Integral Images can reduce the calculation. Instead of calculating at every pixel, it creates the sub-rectangles, and the array references those sub-rectangles and calculates the Haar Features.

$$s(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$

Figure 2: Integral image calculation formula

where: i is pixel matrix, s is integral matrix.

In Figure 3, entry (3,3) in the integral imagine = 1 + 2 + 3 + 4 + 2 + 3 = 15

1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

((a)) Pixel Matrix.

1	3	5	9	10
4	10	13	22	25
6	15	21	32	39
10	20	31	46	59
16	29	42	58	74

((b)) Integral Image.

Figure 3: Pixel Matrix versus Integral Image.

But among all these features we calculated, most of them are irrelevant. The only important features are those of an object, and mostly all the remaining Haar features are irrelevant in the case of object detection. But how do we choose from among the hundreds of thousands of Haar features the ones that best reflect an object?

3.1.3 Adaboost Training

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. (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. 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.

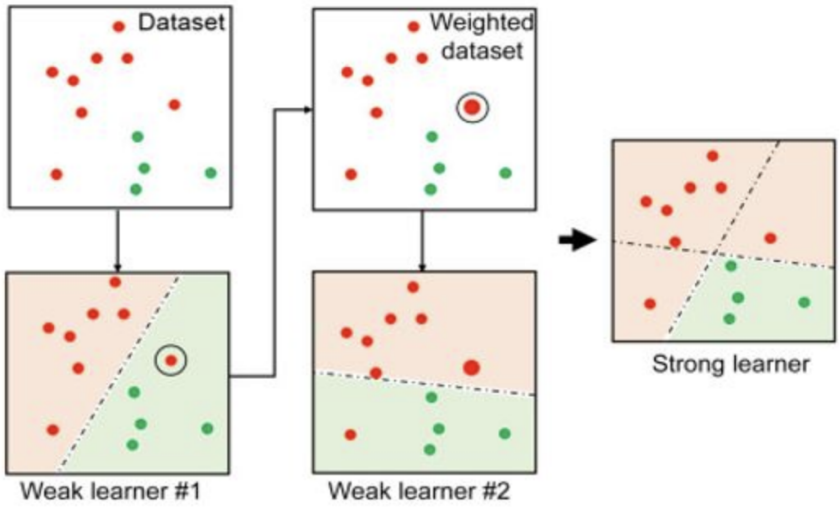


Figure 4: Adaboost Training illustration

3.1.4 Cascading Classifiers

The image is divided into small parts. We put N no of detectors in a cascading manner where each learns a combination of different types of features from images are passed through. Supposedly when the feature extraction is done each sub-part is assigned a confidence value. Sub-images with the highest confidence are detected as face and are sent to the accumulator while the rest are rejected. Thus the cascade fetches the next frame/image if remaining and starts the process again.

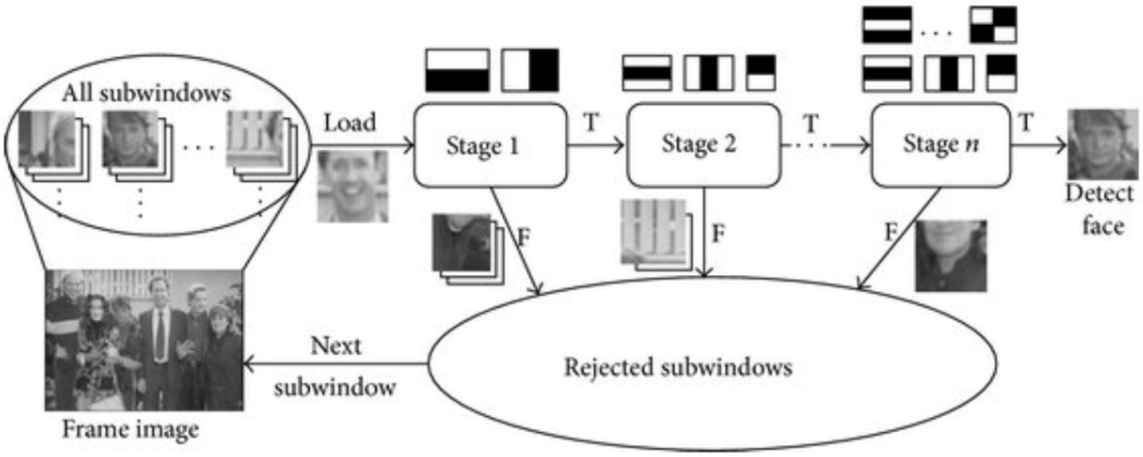


Figure 5: Step to implementing Cascading Classifier

3.2 Face recognition with Local Binary Pattern Histogram (LBPH)

Local Binary Pattern Histogram (LBPH) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. The Local Binary Pattern is used for face

recognition, which means identifying the captured image against the image already stored in the database. The algorithm makes use of four main parameters to recognize a face. The Local Binary Pattern is applied to the image and compared against the central pixel of the image, then we calculate the histogram value for the image.

3.2.1 Compute local binary patterns (LPH)

A formal description of the LBP operator can be given as:

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)$$

,with (x_c, y_c) as central pixel with intensity i_c and i_n being the intensity of the the neighbor pixel. s is the sign function defined as:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$

To understand more about LBP, consider the following example:

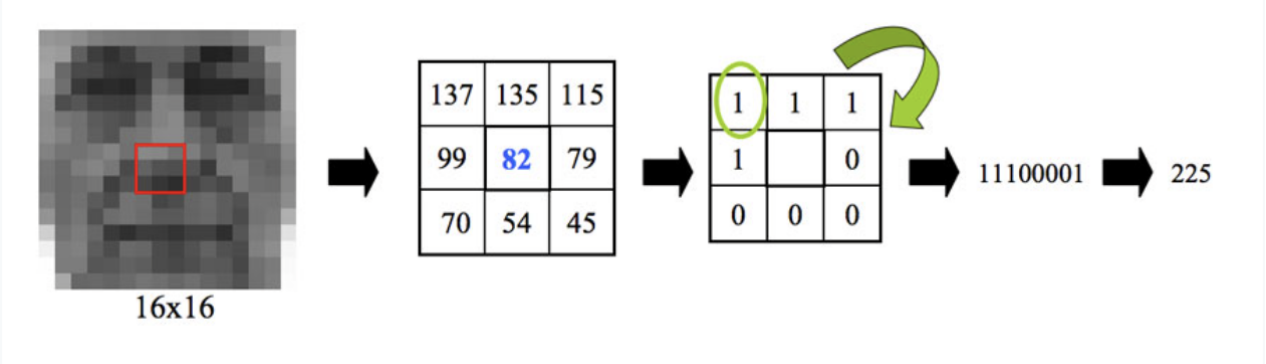


Figure 6: Example of computing LBP

In Figure 7, as we can see the LBP operator is robust against monotonic grayscale transformations.

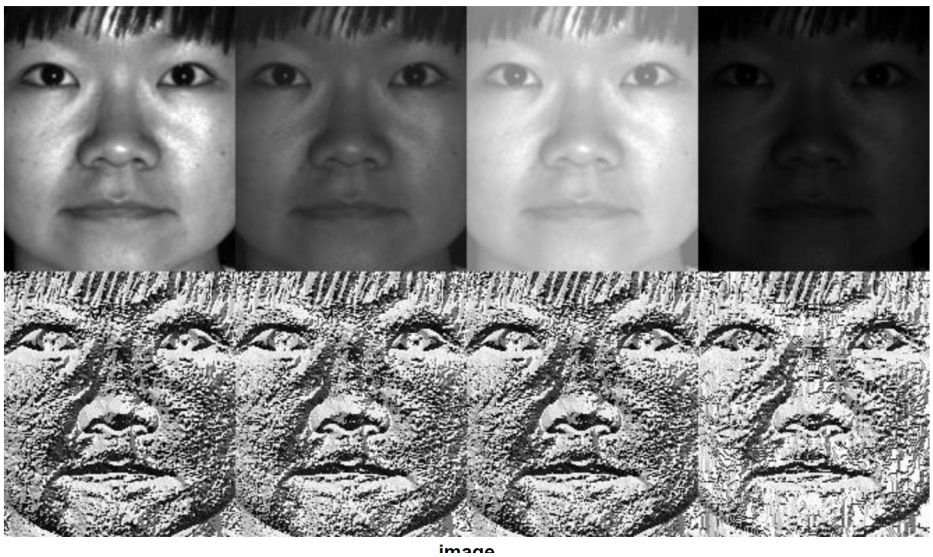


Figure 7: LBP image of an artificially modified image

3.2.2 Create feature vector of the image

After the generation of the LBP value, the number of appearances of LBP codes in the image is put together to form a histogram. After creation of a histogram for each region, all the histograms are merged to form a single histogram and this is known as the feature vector of the image.

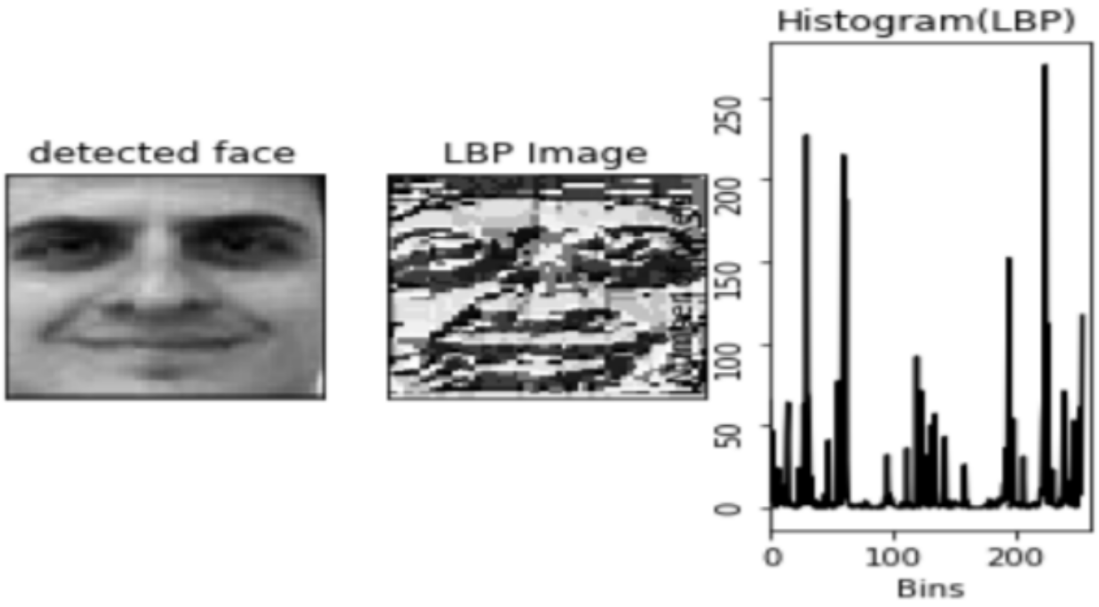


Figure 8: Create Histogram (LBP)

3.2.3 Find matching images in the dataset

We compare the histograms of the test image and the images in the database and then we return the image with the closest histogram. This can be done using many

techniques like Euclidean distance, chi-square, absolute value, etc.

$$d(x,\ y)\ =\ \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

Figure 9: Euclidean distance formula

3.3 Tracking with Kernel Correlation Filter

In our face recognition and tracking model, we use the Kernelized Correlation Filter(KCF) to keep track of detected faces through video frames. In this section, we won't going into abundant mathematical details.

3.3.1 Initialization

The tracking process starts by selecting an initial bounding box around the target face. This is done after the face is detected in the first frame using our face detection algorithm. Once the target is identified, we extract features from this region. These features can include raw pixel values or HOG (Histogram of Oriented Gradients) features.

3.3.2 Training phase

To train the correlation filter, we create a set of training samples by applying various translations to the target patch. This involves generating multiple shifted versions of the target within the initial bounding box. Using these samples, we compute the correlation filter, designed to produce a high response at the target location and low responses elsewhere.We leverage the kernel trick to map the data into a high-dimensional feature space without explicitly computing the coordinates in that space.
The optimal linear filter w is found by solving the regularized least squares problem

$$\min_{w}\Big(||Xw - y||^2 + \lambda||w||^2\Big),$$

where X is a circulant matrix of the target image patch. The rows of X store all possible cyclic image shifts, y is the desired response, λ is a weight of the regularizer.

3.3.3 Tracking Phase

In subsequent frames, we extract features from the region surrounding the last known target location. We then apply a pre-computed correlation filter to these features to obtain a response map The peak in the response map indicates the new location of the target. The model is updated with the new appearance of the target to adapt to changes in the target's appearance over time. This iterative updating process ensures the tracker remains robust against variations in lighting, pose, and partial occlusions.

3.3.4 Handling Scale Variations

To handle changes in the target's scale, we implement a multi-scale search strategy. This involves searching for the target at multiple scales and selecting the scale that yields the highest response. This approach allows the tracker to adjust the size of the bounding box dynamically, ensuring accurate tracking despite changes in the target's distance from the camera.

3.3.5 Optimization Techniques

We utilize several optimization techniques to enhance the efficiency and accuracy of the KCF tracker:

Fourier Transform: We use the Fourier transform to speed up convolution operations. By exploiting the properties of circulant matrices, we can perform these computations efficiently in the frequency domain.
Regularization: Regularization techniques are applied to prevent overfitting, ensuring that the filter generalizes well to new frames. This is crucial for maintaining robust tracking performance across different conditions.

4 Implementation and Results

4.1 Implementation

4.1.1 Data collection and preprocessing phase

We use webcam and the Haar Cascade algorithm from OpenCV to generate the dataset. (Haar Cascade powered by Opencv is pre-trained for detection on large datasets to detect faces in different lighting conditions and angles) Specifically, we only take the part of the face detected by Haar Cascade through the webcam as data for the dataset. For each object, we collect 35 images from different angles. Each captured image is then labeled, converted to grayscale, and resized to the same size. This process enhances the stability and consistency of the model. This process enhances the stability and consistency of the model.

Figure 10: Dataset

4.1.2 Training phase

Firstly, we initialize the object creation of the LBPHFaceRecognizer class from the Opencv library, which is designed to detect faces using the Local Binary Pattern Histogram (LBPH) method. Then, we train based on the processed dataset and the configuration parameters of the trained model are saved into an XML file.

4.1.3 Face Detection and Recognition Phase

In this phase, we continuously read frames from the video webcam. For In this phase, we continuously read frames from the video webcam. For each frame, we convert the image to grayscale and apply the Haar Cascade. If a face is detected, we use the LBPH recognizer to identify the face.

4.1.4 Tracking phase

When a face has been recognized, we initialize the KCF tracker with the bounding box of the detected face. In subsequent frames, the tracker updates the face's position and maintains the bounding box around it. If the tracker loses the face, we return to the recognition phase to detect and re-identify the face.

4.2 Results

When a face is detected and recognized by the trained model, it will annotate the name onto the bounding box as depicted in the image below:

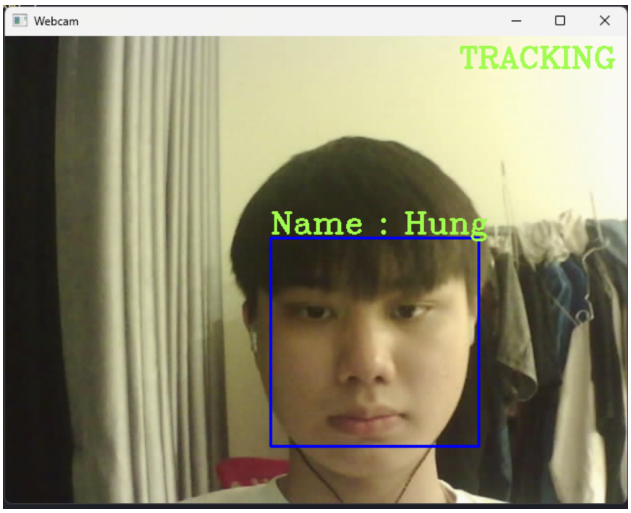
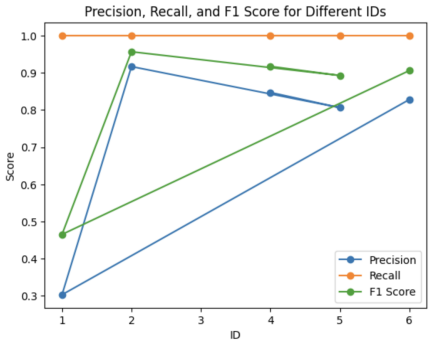


Figure 11: Successful recognition

We tested on 6 registered faces and got the following results:




```
Average Precision: 0.7402369541429944
Average Recall: 1.0
Average F1 Score: 0.8275391737392731
```

Figure 12: Haar Features.

References

[1] OpenCV. Tutorial: Face Detection using Haar Cascades. `https://docs.opencv.org/4.x/da/d60/tutorial_face_main.html`.
[2] Yadav, S.; Payandeh, S. Critical Overview of Visual Tracking with Kernel Correlation Filter. Technologies 2021, 9, 93.
[3] International Journal of Research Publication and Reviews. Vol 3, no 4, 2395-2398, April 2022.
[4] LBPH Algorithm for Face Recognition. `https://iq.opengenus.org/lbph-algorithm-for-face-r`