VIETNAM AVIATION ACADEMY

Department of Telecommunication - Electronics Engineering Technology

LOCATED IN HO CHI MINH CITY

Graduation Thesis

"DROWSINESS DETECTION AND ALERT SYSTEM IN THE CAR"

Written by

Nguyen Van Anh Tuan

Roll.No.1753020018

Under the guidance of

Msc.Vo Phi Son

May 29, 2021

VIETNAM AVIATION ACADEMY

Department of Telecommunication - Electronics Engineering Technology

LOCATED IN HO CHI MINH CITY

Graduation Thesis

"DROWSINESS DETECTION AND ALERT SYSTEM IN THE CAR"

Written by

Nguyen Van Anh Tuan

Roll.No.1753020018

Under the guidance of

Msc.Vo Phi Son

May 29, 2021

PREAMBLE

In nowaday, along with the continuous development and progress of science and technology, image processing is one of the topics that need attention and development. From the first researches about black-white image, gray-scale and digital image, image processing has been studied deeply and applied a lot in our life. Beside that, along with the development of Raspberry Pi with small scale, its promoting more development and application with practice.

The application of Raspberry Pi in image processing aims to provide a few of image processing solutions to apply in real life. In this project, i have used Raspberry Pi to detect drowsiness in the car with algorithms that can respond in real time, the optimal solutions are simple but bring efficiency and high accuracy. I started to identify directly through a camera connected to Raspberry Pi, and programmed using Python with the ability to track and mark the subject's eyes, thereby determining whether the subject was closed or opened and alert a driver immediately, eyes are regconized by the Facial Landmarks algorithm, then calculate the distance between the eyelids using Euclid to detect eye states and detect drowsiness.

Auth.Nguyen Van Anh Tuan

WORDS OF THANKS

Reality show that success is always associated with support of friends, teacher,... And i have special thanks to Mr.Vo Phi Son and my close friends for helping me completing this project.

I have tried my best to do this project. However, due to my lack of experience and knowledge, there are still some unexpected mistakes in the project. Please let me know your opinions and criticizes. Once again, thank you so much.

Auth.Nguyen Van Anh Tuan

CONTENTS

List of Figures

Chapter 1

OVERVIEW ABOUT PROJECT

1.1 Introduction

Nowaday along with the strong development of Science Technology, Robot, Self-Driving Car, AI,... In addition, image processing is a relatively new science compared to many other sciences, but now it is one of the rapidly growing fields and attracts special attention from researchers, research centers, application on this fascinating field. Image processing plays an important role in many practical applications of science and technology as well as in everyday life such as: production and quality assurance, movement of robot, self-driving car, guild tool for the blind, security and monitoring,...

Recently, the popularity and efficiency of using Raspberry Pi kit in applications in science and technology, with characteristics like a miniature computer about the size of a mobile phone, runs an open operating system, is equiped with a powerful processor, low power consumption, and low cost, allowing you to configure the Raspberry Pi kit as a problem-solving computer.

Besides, from the actual needs, drowsiness while driving is quite common and it is also one of the casues of serious accidents, requiring a device that can monitor the state of the person while driving to be able to promptly warn the driver when the driver accidentally falls asleep while driving.

From these reasons has prompted me to research application of Raspberry Pi kit to image processing in order to offer some image processing solutions that can be applied in life.

1.2 Target and The Limits of Project

This project is the first step to learn about the application of processed images in reality, at the same time is also a step to deploy the learned knowledge. Through research and serious work to practice manners, as well as perfecting methods, researching thinking and solving a problem. With the objectives of the project is:

- Learning about Raspberry Pi 3 model B+ kit
- Install OS for Raspberry Pi 3 B+
- Learn about image processing
- Learn about OpenCV, Python

- Install library for OpenCV, Dlib
- Recognize techniques
- Drowsiness Detection by using camera connect to Raspberry Pi and alert to driver through speaker
- Write program
- Experimental model
- Write report

The limit of project is the distance from camera to object from 0,3-1m, detected object not to use glasses and the angle is smaller than 40 degrees, if the object is out of this range, the detection maybe inaccurate or undetectable.

Chapter 2

THEORETICAL BASIS

2.1 Overview About Image Processing

2.1.1 Introduction about Image Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following 3 steps:

- Importing the image via image acquisition tools
- Analysing and manipulating the image
- Output in which result can be altered image or report that is based on image analysis

There are two types of methods used for image processing namely, analogue and digital image processing. Analog image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

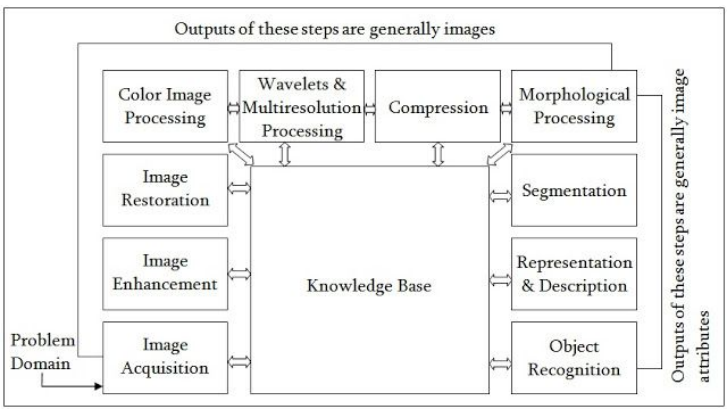

Figure 2.1: Fundamental steps in digital processing

2.1.1.1 Image Acquistion

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling etc.

2.1.1.2 Image Enhancement

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

2.1.1.3 Image Restoration

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

2.1.1.4 Color Image Processing

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modeling and processing in a digital domain etc.

2.1.1.5 Wavelets and Multiresolution Processing

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

2.1.1.6 Compression

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

2.1.1.7 Morphological Processing

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

2.1.1.8 Segmentation

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

2.1.1.9 Representation and Description

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

2.1.1.10 Object Recognition

Recognition is the process that assigns a label, such as, "vehicle" to an object based on its descriptors.

2.1.1.11 Knowledge Base

Knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications.

2.1.2 The Components of Image Processing

2.1.2.1 Digital Image

A digital image is a finite set of pixels with a gray level suitable for describing an image close to the real image. The number of pixels determines the resolution of the image. The higher quality of the image, the more clearly the image's points are displayed, making the image more realistic and sharp.

2.1.2.2 Picture Element

In digital imaging, pixel, pel, or picture element is a smallest addressable element in a raster image, or the smallest addressable element in an **all points addressable display device**; so it is the smallest controllable element of a picture represented on the screen.

Each pixel is a sample of an original image; more samples typically provide more accurate representations of the original. The intensity of each pixel is variable. In color imaging systems, a color is typically represented by three or four component intensities such as red, green, and blue, or cyan, magenta, yellow, and black.

In some contexts (such as descriptions of **camera sensors**), pixel refers to a single scalar element of a multi-component representation (called a photosite in the camera sensor context, although sensel is sometimes used), while in yet other contexts it may refer to the set of component intensities for a spatial position.

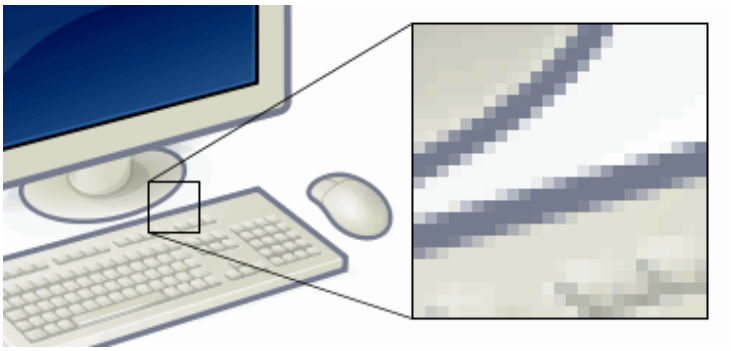

Figure 2.2: Pixel example

Pixel is an element of digital image at coordinate (x,y) with gray level or certain color. The size and the distance between those pixels are chosen appropriately so that the human eye perceives spatial continuity and gray level (or color) of digital image like real image. Each of element in matrix is called an image element.

2.1.2.3 Gray Level of Picture

Gray level is the result of conversion of 1 luminosity value of 1 pixel positive integer value. Usually identified in [0,255] depending on the value each pixel is represented. Common gray scale values is: 16, 32, 64, 128, 256 (level 256 is universal level). The reason is computer techniques use 1 byte (8 bits) to represent the gray level. Gray level use 1 byte represent: 28 = level 256, it mean from 0 to 255).

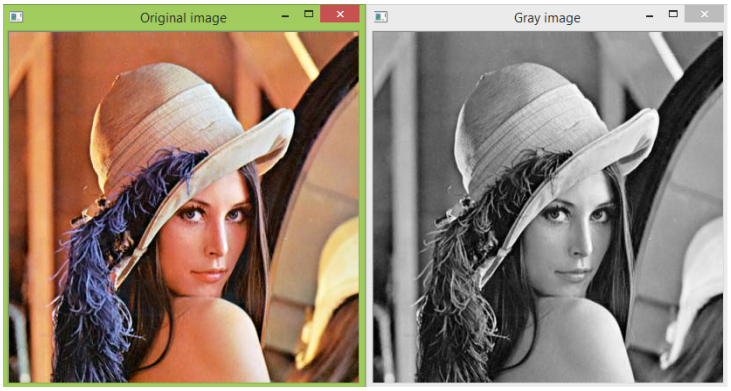

Figure 2.3: Gray scale image example

2.1.2.4 Image Resolution

Image resolution is detail an image holds. The term applies to raster digital images, film images, and other types of images. Higher resolution means more image detail.

Image resolution can be measured in various ways. Resolution quantifies how close lines can be to each other and still be visibly resolved. Resolution units can be tied to physical sizes (e.g. lines per mm, lines per inch), to the overall size of a picture (lines per picture height, also knnown simply as lines, TV lines, or TVL), or to angular subtense. Line pairs are often used instead of lines; a line pair comprises a dark line and an adjacent light line. A line is either a dark line or a light line. A resolution of 10 lines per milimeter means 5 dark line alternating with 5 light lines, or 5 line pairs per

milimeter (5 LP/mm). Photographic lens and film resolution are most often quoted in line pairs per milimeter.

For example: Image resolution in CGA display (Color Graphic Adaptor) is a grid of points across the screen: 320 vertical points * 200 image points (320*200). Obviously, with the same CGA display 12 inches we notice smoother than the screen CGA 17 inches with resolution is 320*200. The reason is with the same resolution but the larger the screen area, the less smooth.

2.1.2.5 Types of image classification

- **Binary Image:** is one that consists of pixels that can have one of exactly two colors, usually black and white. Binary images are also called bi-level or two-level, Pixelart made of two colours is often referred to as 1-Bit or 1bit. This means that each pixel is stored as a single bit—i.e., a 0 or 1.

  The names black-and-white, B&W, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images. In Photoshop parlance, a binary image is the same as an image in "Bitmap" mode.

  Binary images often arise in digital image processing as masks or thresholding, and dithering. Some input/output devices, such as laser printers, fax machines, and bilevel computer displays, can only handle bilevel images.

  A binary image can be stored in memory as a bitmap, a packed array of bits. A 640×480 image requires 37.5 KiB of storage. Because of the small size of the image files, fax machine and document management solutions usually use this format. Most binary images also compress well with simple run-length compression schemes.

  Binary images can be interpreted as subsets of the two-dimensional integer lattice Z^2; the field of morphological image processing was largely inspired by this view.

- **RGB Image:** RGB Color Model is an additive color model, in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

  RGB is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements (such as phosphors or dyes) and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus an RGB value does not define the same color across devices without some kind of color management.

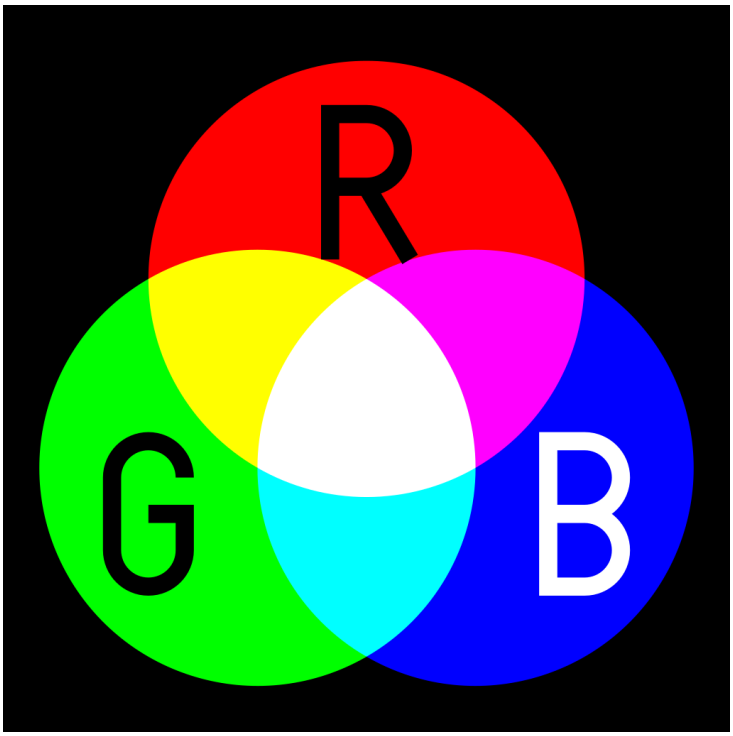

Figure 2.4: Additive color mixing

The choice of primary colors is related to the physiology of the human eye; good primaries are stimuli that maximize the difference between the responses of the cone cells of the human retina to light of different wavelengths, and that thereby make a large color triangle.

The normal three kinds of light-sensitive photoreceptor cells in the human eye (cone cells) respond most to yellow (long wavelength or L), green (medium or M), and violet (short or S) light (peak wavelengths near 570 nm, 540 nm and 440 nm, respectively). The difference in the signals received from the three kinds allows the brain to differentiate a wide gamut of different colors, while being most sensitive (overall) to yellowish-green light and to differences between hues in the green-to-orange region.

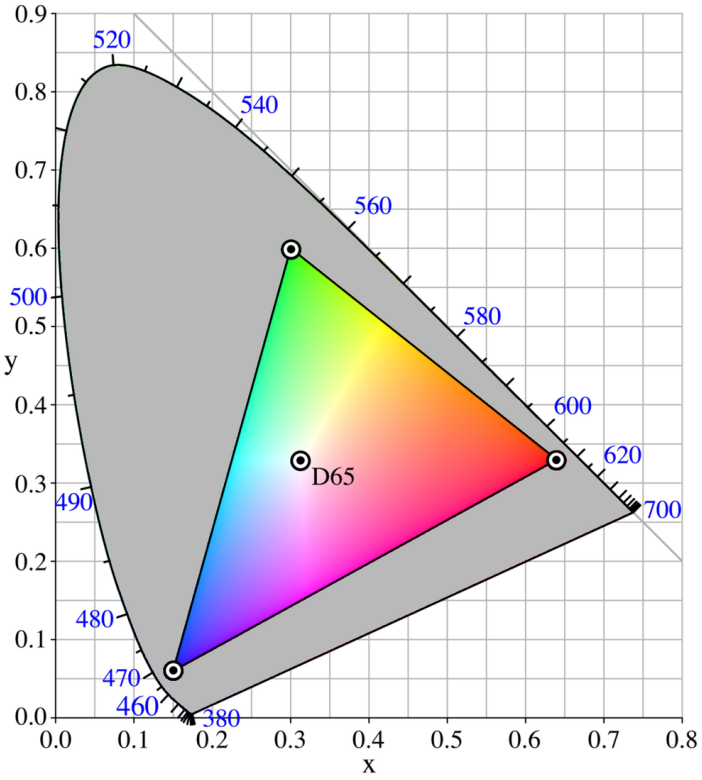

Figure 2.5: A set of primary colors, such as the sRGB primaries, define a color triangle

- **Image Transformation:** is a function. A function that maps one set to another set after performing some operations.

  Image transformation is consider this equation:

$$G(x, y) = Tf(x, y) \tag{2.1}$$

  In this equation, $F(x, y)$ is input image on which transformation function has to be applied; $G(x, y)$ is the output image or processed image; T is the transformation function. This relation between input image and the processed output image can also be represented as: $s = T(r)$ where r is actually the pixel value or gray level intensity of $f(x, y)$ at any point. And s is the pixel value or gray level intensity of $g(x, y)$ at any point.

  The basic gray level transformation has been discussed in our tutorial of basic gray level transformations. There is some image transformations like: `Fourier Transform, Cousin, Sin, convolution transform, Kronecker product`.

2.1.3 Parts of The Image Processing System

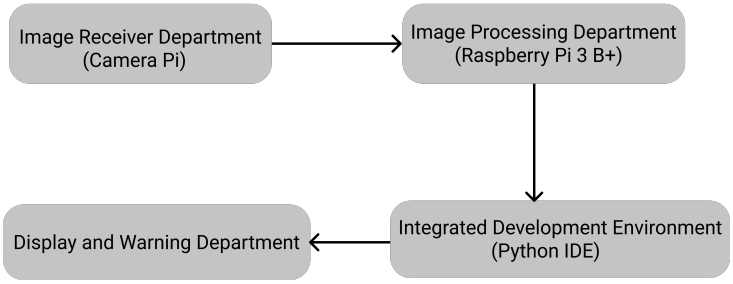

Figure 2.6: The part of image processing system

Image Receiver Department is usually a camera, scaners, image sensor,... In this project, a pi camera with 5mpx resolution is used to capture images.

Image Processing Department is specialized processing equipment or computers,... Specifically here using a Raspberry pi 3B + computer for image processing.

Integrated Development Environment using Thony Python IDE software to write program.

Warning Devices speaker alarms.

2.2 Face Regconition Algorithm

Before we go to the algorithms for face detection we should understand how to detect a face even though we don't know who the subject is.

Face Recognition is a way of recognizing a human face through technology. A facial recognition system uses biometrics to map facial features from a photograph or video. It compares the information with a database of known faces to find a match. Facial recognition can help verify personal identity, but it also raises privacy issues.

The recognition of a face in a video sequence is split into three primary tasks: Face Detection, Face Prediction, and Face Tracking. The tasks performed in the Face Capture program are performed during face recognition as well. To recognize the face obtained, a vector of HOG features of the face is extracted. This vector is then used in the SVM model to determine a matching score for the input vector with each of the labels. The SVM returns the label with the maximum score, which represents the confidence to the closest match within the trained face data.

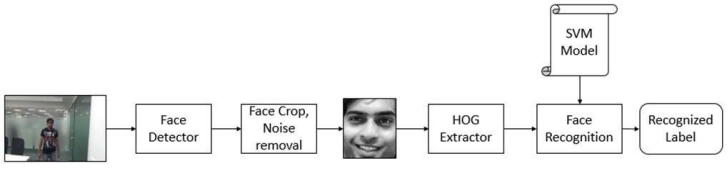

Figure 2.7: Block diagram of the face recognition process

The task of calculating matching scores is exceptionally heavy to compute. Hence, once detected and identified, the labeled face in an image needs to be tracked to reduce the computation in future frames until the face eventually disappears from the video. Of all

the available trackers, the Camshift tracking algorithm is used since it produces the best results with faces.

Where you see a face, recognition technology sees data. That data can be stored and accessed. For instance, half of all American adults have their images stored in one or more facial-recognition databases that law enforcement agencies can search, according to a Georgetown University study. Technologies can be different, but there are the basic steps:

- **Step 1.** A picture of your face is captured from a photo or video. Your face might appear alone or in a crowd. Your image may show you looking straight ahead or nearly in profile
- **Step 2.** Facial recognition software reads the geometry of your face. Key factors include the distance between your eyes and the distance from forehead to chin. The software identifies facial landmarks — one system identifies 68 of them — that are key to distinguishing your face. The result: your facial signature
- **Step 3.** Your facial signature — a mathematical formula — is compared to a database of known faces. And consider this: at least 117 million Americans have images of their faces in one or more police databases. According to a May 2018 report, the FBI has had access to 412 million facial images for searches
- **Step 4.** A determination is made. Your faceprint may match that of an image in a facial recognition system database.

The gist of the pipeline can be seen in figure down here:

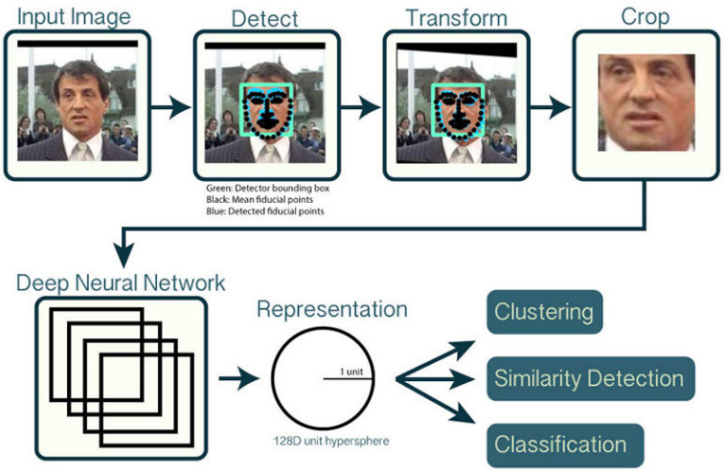

Figure 2.8: An overview of the OpenCV face recognition pipeline

First, we input an image or video frame to our face recognition pipeline. Given the input image, we apply face detection to detect the location of a face in the image. Optionally we can compute **Facial Landmarks**, enabling us to **Preprocess and align the face**.

Face alignment, as the name suggests, is the process of identifying the geometric structure of the faces and attempting to obtain a canonical alignment of the face based on translation, rotation, and scale. While optional, face alignment has been demonstrated to increase face recognition accuracy in some pipelines. After we've (optionally) applied face alignment and cropping, we pass the input face through our deep neural network:

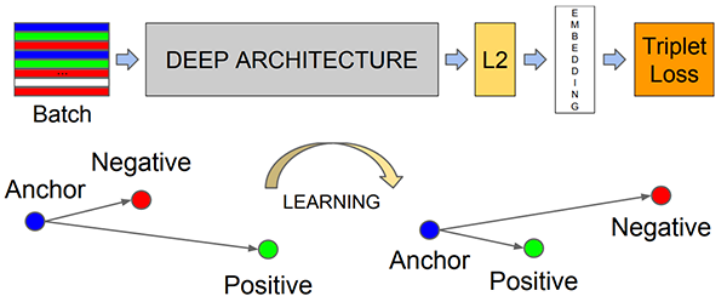

Figure 2.9: How the deep learning face recognition model computes the face embedding

The FaceNet deep learning model computes a 128-d embedding that quantifies the face itself. But how does the network actually compute the face embedding? The answer lies in the training process itself, including:

- The input data to the network
- The triplet loss function

To train a face recognition model with deep learning, each input batch of data includes three images:

- The anchor
- The positive image
- The negative image

The anchor is our current face and has identity A.

The second image is our positive image — this image also contains a face of person A.

The negative image, on the other hand, does not have the same identity, and could belong to person B, C, or even Y!

The point is that the anchor and positive image both belong to the same person/face while the negative image does not contain the same face. The neural network computes the 128-d embeddings for each face and then tweaks the weights of the network (via the triplet loss function) such that:

- The 128-d embeddings of the anchor and positive image lie closer together
- While at the same time, pushing the embeddings for the negative image father away

In this manner, the network is able to learn to quantify faces and return highly robust and discriminating embeddings suitable for face recognition.

2.2.1 Face Detection using HOG

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this

value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

In the current example, all the face sample images of a person are fed to the feature descriptor extraction algorithm; i.e., a HOG. The descriptors are gradient vectors generated per pixel of the image. The gradient for each pixel consists of magnitude and direction, calculated using the following formular:

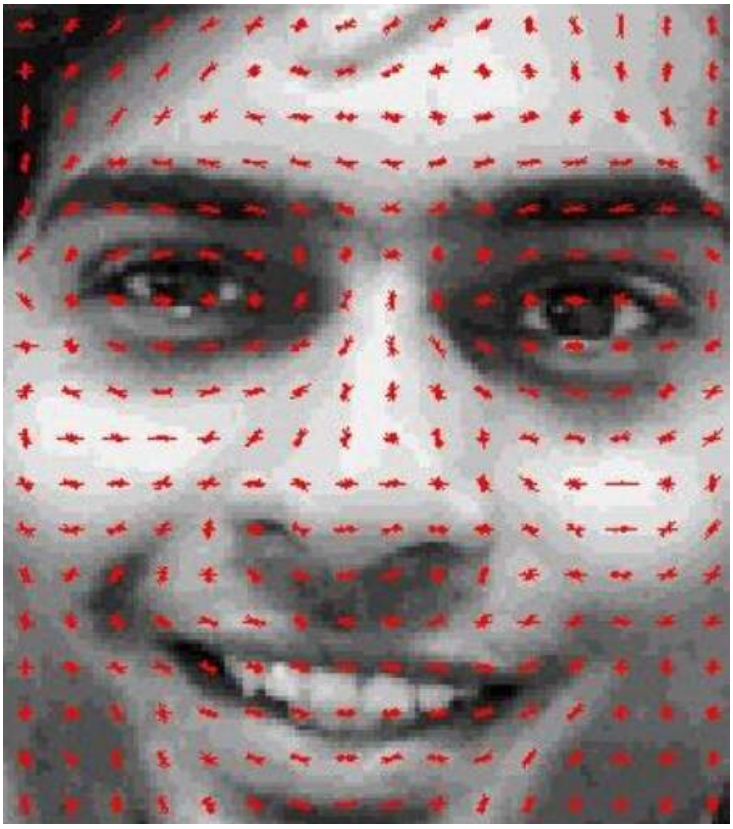

Figure 2.10: HOG features sample face

$$g = \sqrt{g_x^2 + g_y^2} \tag{2.2}$$

$$\theta = \arctan \frac{g_y}{g_x} \tag{2.3}$$

Gx and Gy are respectively the horizontal and vertical components of the change in the pixel intensity. A window size of 128 x 144 is used for face images since it matches the general aspect ratio of human faces. The descriptors are calculated over blocks of pixels with 8 x 8 dimensions. These descriptor values for each pixel over 8 x 8 block are quantized into 9 bins, where each bin represents a directional angle of gradient and value in that bin, which is the summation of the magnitudes of all pixels with the same angle.

Further, the histogram is then normalized over a 16 x 16 block size, which means four blocks of 8 x 8 are normalized together to minimize light conditions. This mechanism mitigates the accuracy drop due to a change in light. The SVM model is trained using a number of HOG vectors for multiple faces.

There is some review entire detailed process of training an object detector using Histogram Oriented Gradients, each step can be fairly detailed. It goes like something

like this:

- **Step 1:** Sample P positive samples from your training data of the object(s) you want to detect and extract HOG descriptors from these samples;
- **Step 2:** Sample N negative samples from a negative training set that **does not contain** any of the objects you want to detect and extract HOG descriptors from these samples as well. In practice $N \gg P$;
- **Step 3:** Train a Linear Support Vector Machine on your positive and negative samples;
- **Step 4:**

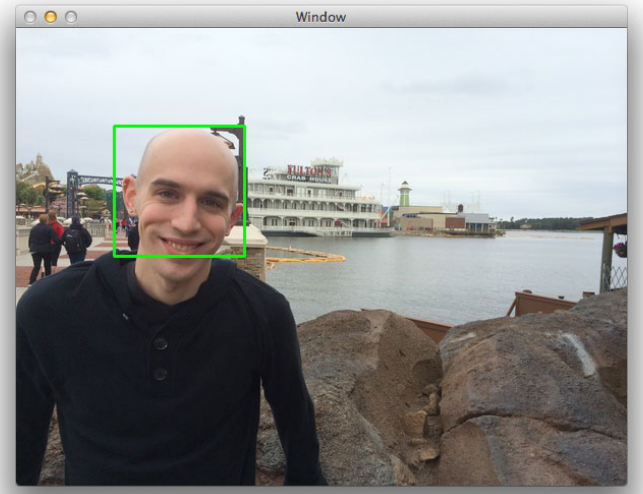

Figure 2.11: Example of the sliding a window approach, where we slide a window from left-to-right and top-to-bottom

Apply hard-negative mining, For each image and each possible scale of each image in your negative training set, apply the sliding window technique and slide your window across the image. At each window compute your HOG descriptors and apply your classifier. If your classifier (incorrectly) classifies a given window as an object (and it will, there will absolutely be false-positives), record the feature vector associated with the false-positive patch along with the probability of the classification. **This approach is called hard-negative mining**.

- **Step 5:** Take the false-positive samples found during the hard-negative mining stage, sort them by their confidence (i.e. probability) and re-train your classifier using these hard-negative samples.
- **Step 6:** Your classifier is now trained and can be applied to your test dataset. Again, just like in Step 4, for each image in your test set, and for each scale of the image, apply the sliding window technique. At each window extract HOG descriptors and apply your classifier. If your classifier detects an object with sufficiently large probability, record the bounding box of the window. After you have finished scanning the image, apply non-maximum suppression to remove redundant and overlapping bounding boxes.

These are the bare minimum steps required, but by using this 6-step process you can train and build object detection classifiers of your own! Extensions to this approach include a deformable parts model and Exemplar SVMs, where you train a classifier for each positive instance rather than a collection of them.

However, if you've ever worked with object detection in images you've likely ran into the problem of detecting multiple bounding boxes around the object you want to detect in the image. And here's an example of this overlapping bounding box problem:

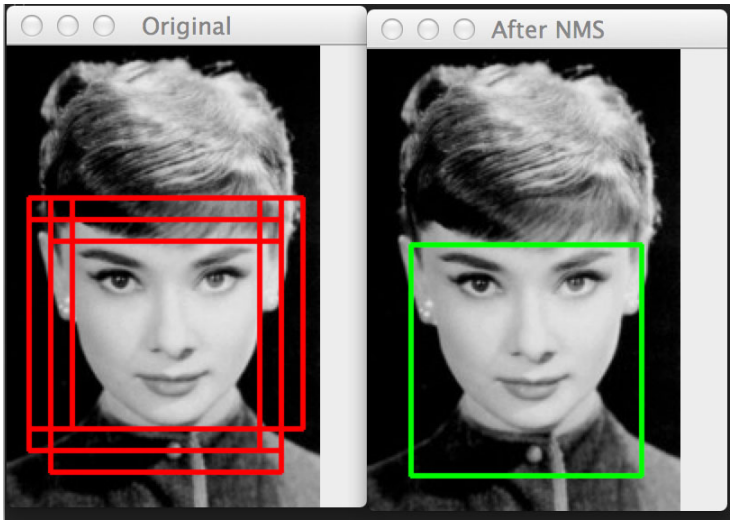

Figure 2.12: (Left) Detecting multiple overlapping bounding boxes around the face we want to detect. (Right) Applying non-maximum suppression to remove the redundant bounding boxes.

2.2.2 Haar-like Feature (Haar-Cascade)

2.2.2.1 Theory

Object Detection using Haar feature-based cascade classifiers is an effective object detection method is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. 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.

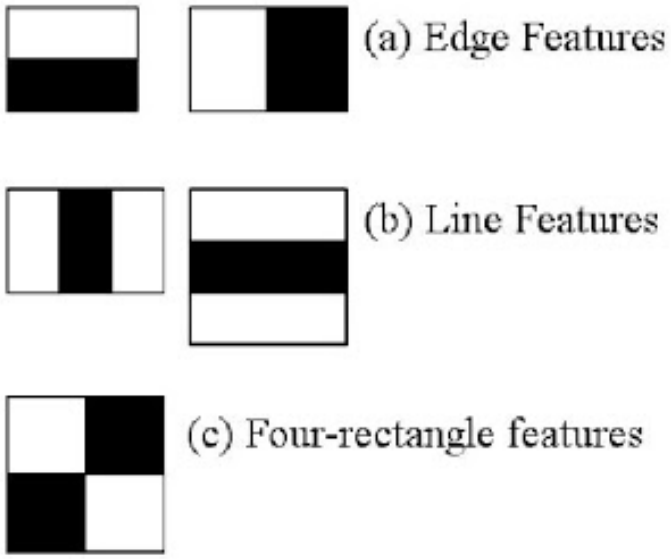

Figure 2.13: Feature 4 rectangle

Figure 2.14: Feature in center

Using above features, we can calculate the value of the Haar-Like feature as the difference between the sum of the pixels of the black area and the white area as shown in the following formula:

$$F(x) = Sumofblackarea - Sumofwhitearea(graylevelofpixel) \tag{2.4}$$

There is a concept called **"Integral Image"**, is the 2D array with the size equal to the size of the image to be Haar-Like feature, with each element of this array is computed by summing the pixels above and left of it.

2.2.2.2 Integral Image

The idea of this concept is transforming an input images into a summed-area table, where the value at any point (x, y) in that table is the sum of all the pixels above and to the left of (x, y), inclusive:

$$P(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.5}$$

Where I(x,y) is the value of the integral image pixel in the position (x,y), while i(x,y) is the corresponding intensity in the original image. It is a recursive formula, hence, if we

start from one corner of the input image, we will have the same result in the integral image. To make it clearer, let's see an example:

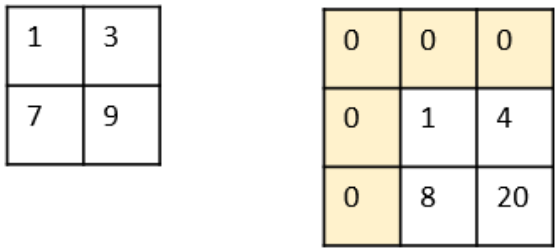

Figure 2.15: Example of Integral Image

In this figure we added one row and column of zeros, since we need one step backward in order to start the recursive formula. Hence, if your image is w pixels wide and h pixels high, then the integral of this will be $w+1$ pixels wide and $h+1$ pixels high.

Moving to the computations, let's start from the first pixel in the original image with intensity 1: the integral image returns exactly the same value, since it is computing (1+0+0). Then, pixel '3' becomes '4', since it is 3+1+0+0. With the same procedure, we obtain an "8" (7+1+0) and a '20' (9+3+1+7).

We have a new image, but how is supposed to be useful? The answer rely in an unique property of the integral image. Indeed, it turned our that if you need to compute the summation within a window in the input image, hence that summation is equal to a linear combination of the corresponding window's corner in the integral image, as follows:

$$\Sigma_{x_0<x<x_1;y_0<y<y_1} i(x,y) = I(D) + I(A) - I(B) - I(C) \tag{2.6}$$

Where is A, B, C and D are the corners of the corresponding window in the integral image.

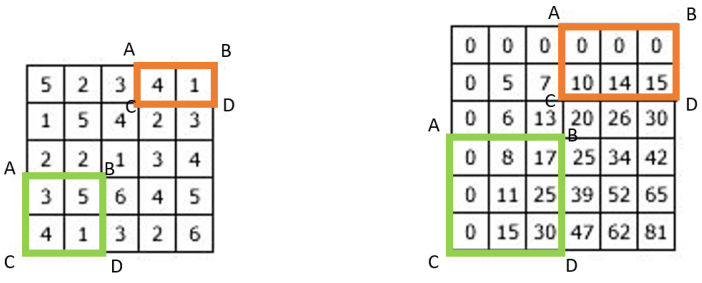

Figure 2.16: Integral Image Approach

This reduces the number of computations by far. To give you an idea, consider a 100×100 image with a 9×9 window. We want to compute the sum of the pixel intensities within that window, which requires 8 operations. If we repeat this procedure 100 times, we obtain 800 operations.

Now let's see the integral image approach. First, we compute the summed-area table, which requires 56 operations. Then, considering the same 9×9 window, to compute the sum of pixel intensity we just need the above formula, which is made of 3 operations. Hence, the total number of operations is 56+3*100=356. As you can see, it is less than a half.

This procedure is widely used in computer vision and Haar Cascade algorithm is based exactly on that.

The idea is passing these filters on the image, inspecting one portion (or window) at the time. Then, for each window, all the pixel intensities of, respectively, white and black portions are summed. Finally, the value obtained by subtracting those two summations is the value of the feature extracted. Ideally, a great value of a feature means it is relevant. Namely, if we consider the Edge feature and apply it to the following Black and White pic:

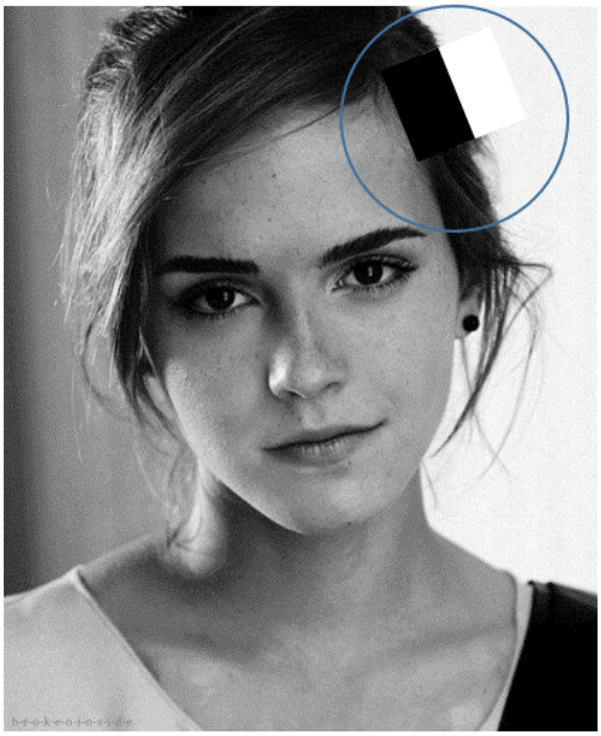

Figure 2.17: Black and White picture

We will obtain a significant value, hence the algorithm will return an edge feature with high probability. Of course, the real intensities of pixels is never equal to white or black, and we will often face a similar situation:

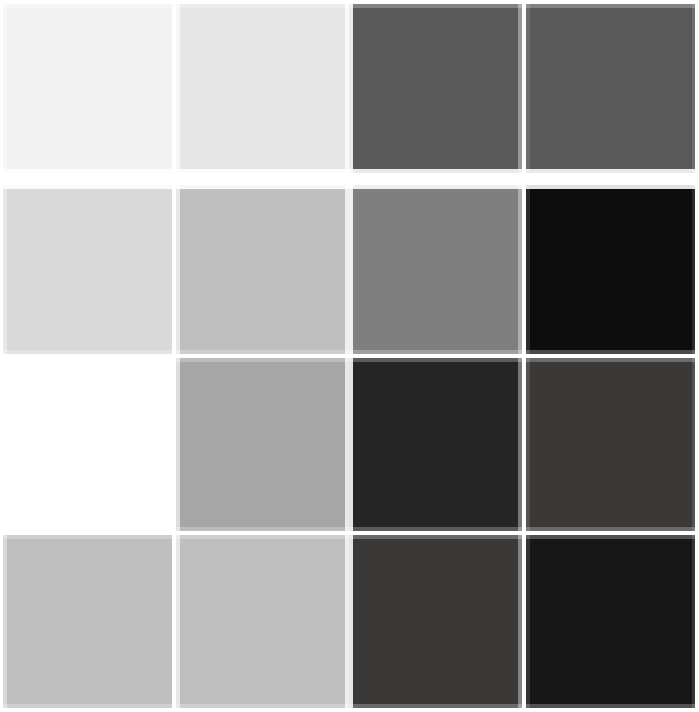

Figure 2.18: The real intensities of pixels

Nevertheless, the idea remains the same: the higher the result (that is, the difference between black and white summations), the higher the probability of that window of being a relevant feature.

Imagine that the huge amount of features returned by this computation. To give you an idea, even a 24x24 window results over 160000 features, and windows within an image are a lot. How to make this process more efficient? The solution came out with the concept of Summed-area table, also known as Integral Image. It is a data structure and algorithm for generating the sum of values in a rectangular subset of a grid. The goal is reducing the number of computations needed to obtain the summations of pixel intensities within a window.

Next is also involves efficiency and optimization. Besides being numerous, features might also be irrelevant. Among the features we obtain (that are more than 160000), how can we decide which ones are good? The answer to this question relies on the concept of Ensembilg method: by combining many algorithms, weak by definition, we can create a strong algorithm. This is accomplished using **Adaboost** which both selects the best features and trains the classifiers that use them. This algorithm constructs a "strong" classifier as a linear combination of weighted simple "weak" classifiers.

The last concept which needs to be introduced is a final element of optimization (in terms of the time of training). Indeed, even though we reduced our 160000+ features to a more manageable number, the latter is still high: applying all the features on all the windows will take a lot of time. That's why we use the concept of Cascade of classifiers: instead of applying all the features on a window, it groups the features into different stages of classifiers and applies one-by-one. If a window fails (the difference between white and black summations is low) the first stage (which normally includes few

features), the algorithm discards it: it won't consider remaining features on it. If it passes, the algorithm applies the second stage of features and continues the process.

2.2.3 AdaBoost Algorithm

In recent years, boosting algorithms gained massive popularity in data science or machine learning competitions. Most of the winners of these competitions use boosting algorithms to achieve high accuracy. These Data science competitions provide the global platform for learning, exploring and providing solutions for various business and government problems. Boosting algorithms combine multiple low accuracy(or weak) models to create a high accuracy(or strong) models. It can be utilized in various domains such as credit, insurance, marketing, and sales. Boosting algorithms such as AdaBoost, Gradient Boosting, and XGBoost are widely used machine learning algorithm to win the data science competitions. In this tutorial, you are going to learn the AdaBoost ensemble boosting algorithm, and the following topics will be covered:

- Ensemble Machine Learning Approach
  - Bagging
  - Boosting
  - Stacking
- Adaboost Classifier
- How does the AdaBoost algorithm work?

2.2.3.1 Ensemble Machine Learning Approach

An ensemble is a composite model, combines a series of low performing classifiers with the aim of creating an improved classifier. Here, individual classifier vote and final prediction label returned that performs majority voting. Ensembles offer more accuracy than individual or base classifier. Ensemble methods can parallelize by allocating each base learner to different-different machines. Finally, you can say Ensemble learning methods are meta-algorithms that combine several machine learning methods into a single predictive model to increase performance. Ensemble methods can decrease variance using bagging approach, bias using a boosting approach, or improve predictions using stacking approach.

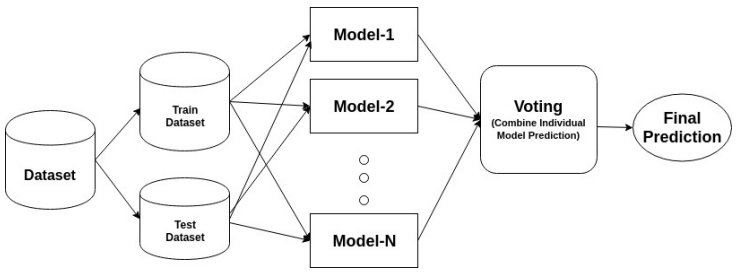

Figure 2.19: Ensemble methods

1. **Bagging** stands for bootstrap aggregation. It combines multiple learners in a way to reduce the variance of estimates. For example, random forest trains M Decision Tree, you can train M different trees on different random subsets of the data and perform voting for final prediction. Bagging ensembles methods are Random Forest and Extra Trees;

2. **Boosting algorithms** are a set of the low accurate classifier to create a highly accurate classifier. Low accuracy classifier (or weak classifier) offers the accuracy better than the flipping of a coin. Highly accurate classifier(or strong classifier) offer error rate close to 0. Boosting algorithm can track the model who failed the accurate prediction. Boosting algorithms are less affected by the overfitting problem. The following three algorithms have gained massive popularity in data science competitions.
    - AdaBoost (Adaptive Boosting)
    - Gradient Tree Boosting
    - XGBoost
3. **Stacking (or stacked generalization)** is an ensemble learning technique that combines multiple base classification models predictions into a new data set. This new data are treated as the input data for another classifier. This classifier employed to solve this problem. Stacking is often referred to as blending.

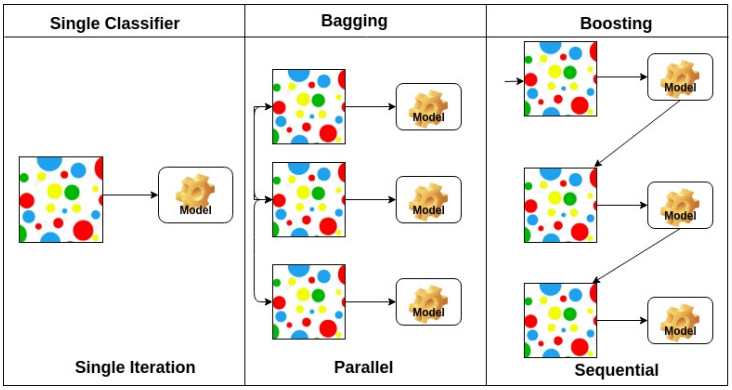

Figure 2.20: On the basis of the arrangement of base learners, ensemble methods can be divided into two groups: In parallel ensemble methods, base learners are generated in parallel for example. Random Forest. In sequential ensemble methods, base learners are generated sequentially for example AdaBoost.

On the basis of the type of base learners, ensemble methods can be divided into two groups: homogenous ensemble method uses the same type of base learner in each iteration. heterogeneous ensemble method uses the different type of base learner in each iteration.

2.2.3.2 AdaBoost Classifier

Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Any machine learning algorithm can be used as base classifier if it accepts weights on the training set. Adaboost should meet two conditions:

1. The classifier should be trained interactively on various weighed training examples;
2. In each iteration, it tries to provide an excellent fit for these examples by minimizing training error.

2.2.3.3 How Does AdaBoost Algorithm Work?

It works in the following steps:

1. Initially, Adaboost selects a training subset randomly;
2. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training;
3. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification;
4. Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight;
5. This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators;
6. To classify, perform a "vote" across all of the learning algorithms you built.

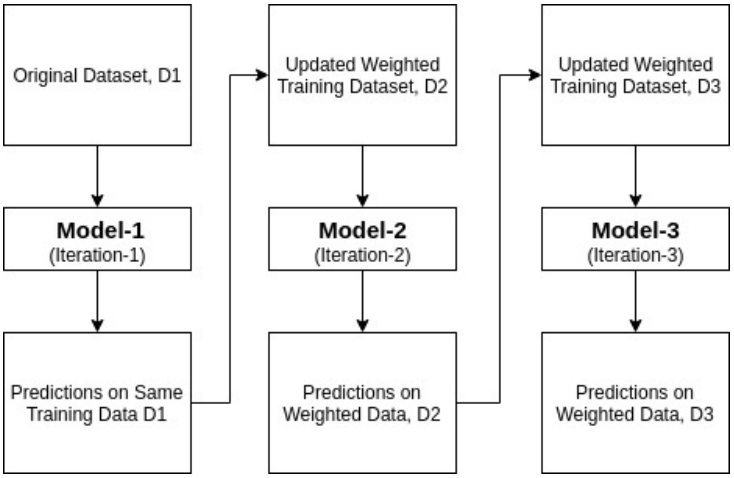

Figure 2.21: Adaboost model work

$$h_k = \begin{cases} 1 & p_k f_k(x) < p_k \theta_k \\ 0 & \text{opposite} \end{cases} \tag{2.7}$$

With:

- x: Subwindow need to consider
- θ: threshold
- f_k: Haar-like characteristic value
- p_k: Decision coefficient of detemining the dimension of the equation

Adaboost will combine weak classifiers into strong classifier as follows:

$$H(x) = \Sigma(a_1 h_1(x) + a_2 h_2(x) + ... + a_n h_n(x)) \tag{2.8}$$

With: $a_t >= 0$ is normalization coefficient for weak classifiers.

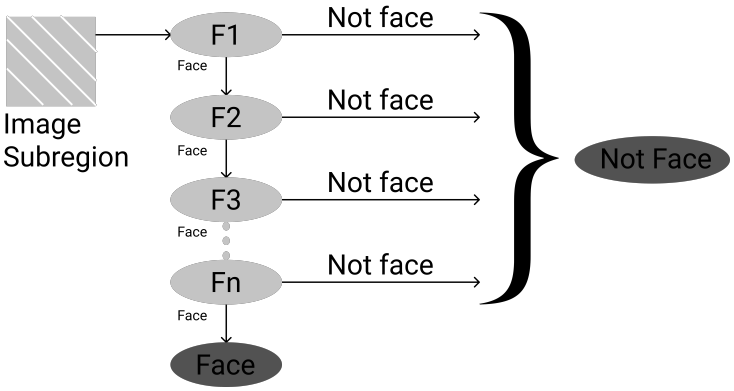

Figure 2.22: Haar Cascade Detection model

2.2.4 Eye Blink Detection

The eye blink is a fast closing and reopening of a human eye. Each individual has a little bit different pattern of blinks. The pattern differs in the speed of closing and opening, a degree of squeezing the eye and blink duration. The eye blink lasts approximately 100-400 ms.

To build blink detector, we'll be computing a metric called the Eye Aspect Ratio (EAR). Unlike traditional image processing methods for computing blinks which typically involve some combination of:

1. Eye localization
2. Thresholding to find the whites of the eyes
3. Determining if the "white" region of the eyes disappears for a period of time (indicating a blink).

The eye aspect ratio is instead a **much more elegant solution** that involves a very **simple calculation** based on the ratio of distances between facial landmarks of the eyes.

2.2.4.1 Understanding the "Eye Aspect Ratio"(EAR)

To apply facial landmark detection to localize important regions of the face, including eyes, eyebrows, nose, ears and mouth.

This also implies that we can extract specific facial structures by knowing the indexes of the particular face parts.

2.2.4.1.1 In terms of blink detection, we are only interested in two sets of facial structures — the eyes Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the remainder of the region:

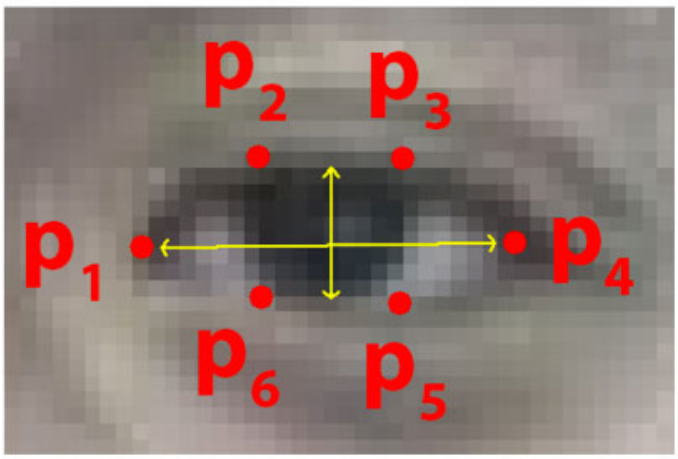

Figure 2.23: The 6 facial landmarks associated with the eye

2.2.4.1.2 There is a relation between the width and the height of these coordinates For every video frame, the eye landmarks are detected. The Eye Aspect Ratio between height and width of the eye is computed:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||} \tag{2.9}$$

Where p1, . . . , p6 are 2D facial landmark locations.

The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since there is only one set of horizontal points but two sets of vertical points.

This equation is so interesting because the eye aspect ratio is approximately constant while the eye is open, but will rapidly fall to zero when a blink is taking place.

Using this simple equation, we can avoid image processing techniques and simply rely on the ratio of eye landmark distances to determine if a person is blinking.

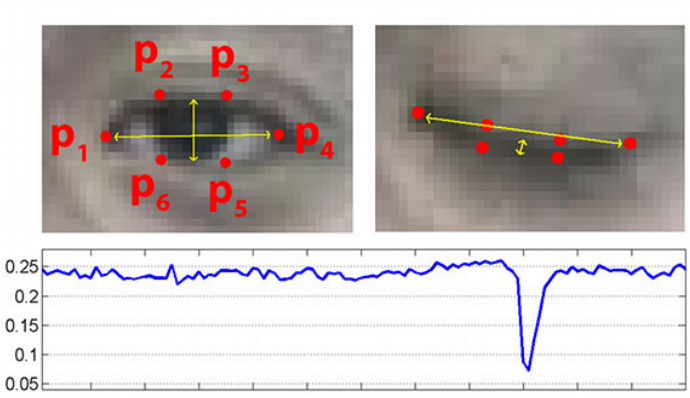

Figure 2.24: Top-left: A visualization of eye landmarks when then the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time

In this figure, we can see that on the top-left we have an eye that is fully open — the

eye aspect ratio here would be large(r) and relatively constant over time. However, once the person blinks (top-right) the eye aspect ratio decreases dramatically, approaching zero. The bottom figure plots a graph of the eye aspect ratio over time for a video clip. As we can see, the eye aspect ratio is constant, then rapidly drops close to zero, then increases again, indicating a single blink has taken place.

2.2.5 Facial Landmarks Algorithm

Facial landmark detection aims to detect the location of predefined facial landmarks, such as the corners of the eyes, eyebrows, the tip of the nose. It has drawn much attention recently as it is a prerequisite in many computer vision applications.For example, facial landmark detection can be applied to a large variety of tasks, including face recognition, head pose estimation, facial reenactment and 3D face reconstruction...

2.2.5.1 What is Facial Landmark?

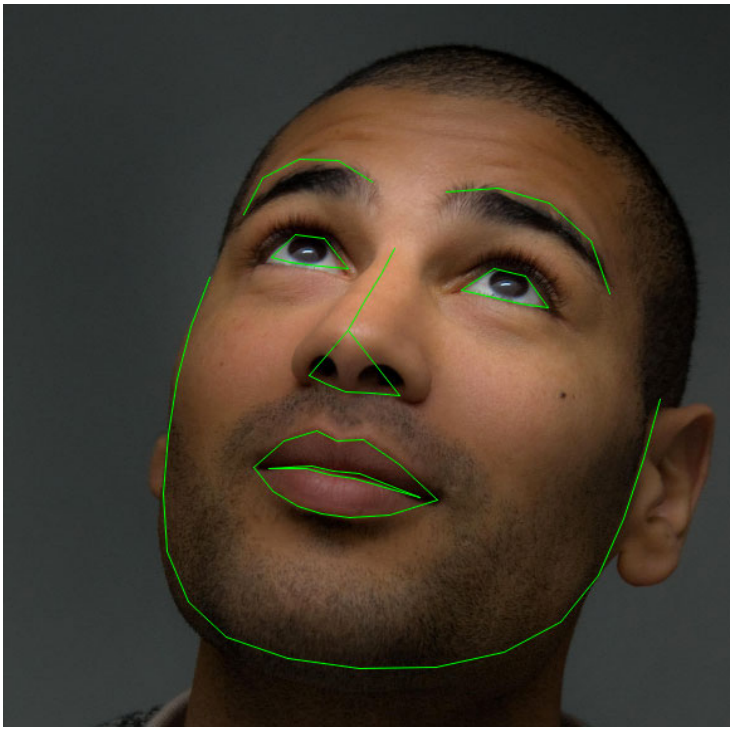

Figure 2.25: Facial landmarks are used to label and identify key facial attributes in an image

Detecting facial landmarks is a **subset of the shape prediction problem**. Given an input image (and normally an **ROI** that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape. Detecting facial landmarks is therefore a two step process:

- **Step #1**: Localize the face in the image;
- **Step #2**: Detect the key facial structures on the face ROI.

Face detection (Step #1) can be achieved with OpenCV's built-in **Haar Cascade**.

We might apply a pre-trained **HOG + Linear SVM Object Detector** specifically for the task of face detection. Or we might even use deep learning-based algorithms for face localization.

The important is that through some method we obtain the face bounding box (i.e., the (x, y)-coordinates of the face in the image).

Given the face recognition we can then apply **Step #2: detecting key facial structures in the face region**. There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

1. Mouth
2. Right, left eyebrow
3. Right, left eye
4. Nose
5. Jaw

This method starts by using:

1. A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure;
2. Priors, of more specifically, the probability on distance between pairs of input pixels.

The end result is a facial landmark detector that can be used to **detect facial landmarks in real-time** with **high quality predictions**.

2.2.5.2 Understanding Dlib's Facial Landmark Detector

The pre-trained 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:

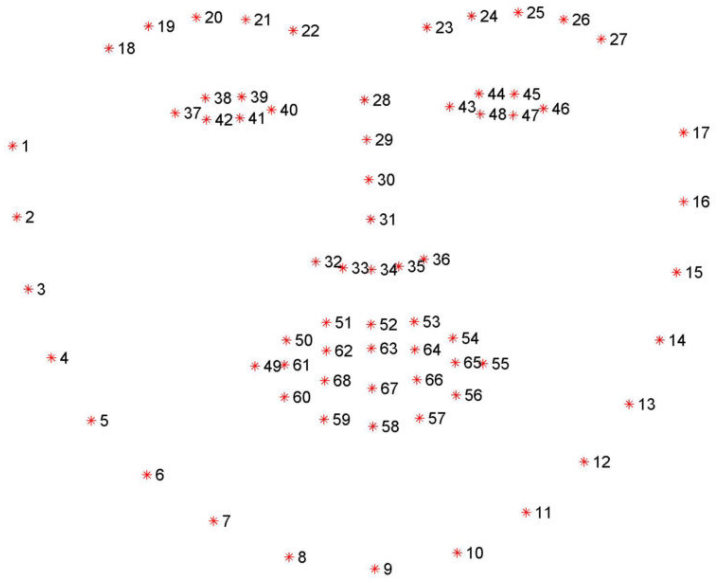

Figure 2.26: Visualizing the 68 facial landmark coordinates

These annotations are part of the 68 points **iBug 300-W dataset** which the dlib facial landmark predictor was trained on.

It's important to note that other flavors of facial landmark detectors exist, including the 194 points model that can be trained on the **HELEN dataset**.

2.3 Euclidean Distance

2.3.1 Definition

In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points. It can be calculated from the Cartesian coordinates of the points using the **Pythagorean theorem**, therefore occasionally being called the **Pythagorean distance**. These names come from the ancient Greek mathematicians Euclid and Pythagoras, although Euclid did not represent distances as numbers.

The distance between two objects that are not points is usually defined to be the smallest distance among pairs of points from the two objects. Formulas are known for computing distances between different types of objects, such as the distance from a point to a line. In advanced mathematics, the concept of distance has been generalized to abstract **metric spaces**, and other distances than Euclidean have been studied. In some applications in statistics and optimization, the square of the Euclidean distance is used instead of the distance itself.

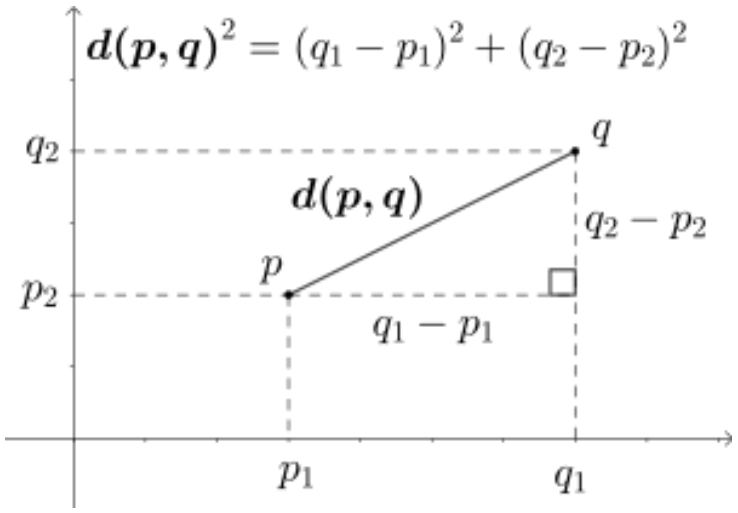

Figure 2.27: Using the Pythagorean theorem to compute two-dimensional Euclidean distance

2.3.2 Distance Formulas

2.3.2.1 One Dimension

The distance between any 2 points on the real line is the absolute value of the numerical difference of their coordinates. Thus if p and q are two points on the real line, then the distance between them is given by:

$$d(p, q) = |p - q| \tag{2.10}$$

A more complicated formula, giving the same value, but generalizing more readily to higher dimensions, is:

$$d(p, q) = \sqrt{(p - q)^2} \tag{2.11}$$

In this formula, squaring and then taking the square root leaves any positive number unchanged, but replaces any negative number by its absolute value.

2.3.2.2 Two Dimensions

In the **Euclidean plane**, let point p have **Cartesian coordinates** (p_1, p_2) and let point q have coordinates (q_1, q_2). The the distance between p and q is given by:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{2.12}$$

This can be seen by applying the **Pythagorean theorem** to a right triangle with horizontal and vertical sides, having the line segment from p to q as its hypotenuse. The two squared formulas inside the square root give the areas of squares on the horizontal and vertical sides, and the outer square root converts the area of the square on the hypotenuse into the length of the hypotenuse.

It is also possible to compute the distance for points given by **polar coordinates**. If the polar coordinates of p are (r, θ) and the polar coordinates of q are (s, ψ), then their distance is:

$$d(p, q) = \sqrt{r^2 + s^2 - 2rs\cos(\theta - \psi)} \tag{2.13}$$

When p and q are the are expressed as **complex number** and **complex plane**, the same formula for one-dimensional points expressed as real numbers can be used:

$$d(p, q) = |p - q| \tag{2.14}$$

2.3.2.3 Higher Dimensions

In three dimensions, for points given by their Cartesian coordinates, the distance is:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \tag{2.15}$$

In general, for points given by Cartesian coordinates in n-dimensional Euclidean space, the distance is:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \tag{2.16}$$

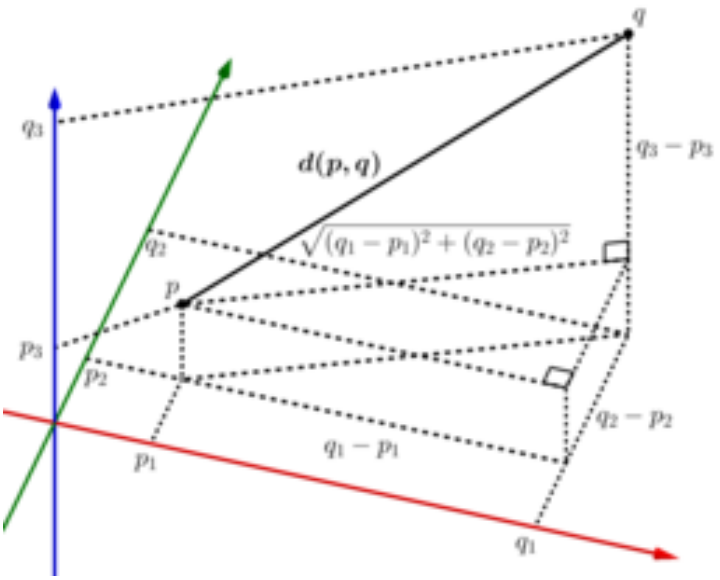

Figure 2.28: Deriving the n-dimensional Euclidean distance formula by repeatedly applying the Pythagorean theorem

2.4 OpenCV

2.4.1 Overview

OpenCV (Open Computer Vision) is a leading open source library for computer vision processing, machine learning and image processing. OpenCV is written in C/C++, so it has very fast computation speed, can be used with real-time related applications. The library has more than 2500 optimized algorithms.

Opencv has interfaces for C / C ++, Python Java, so it supports Windows, Linux, MacOs and Android, iOS OpenCV has a community of more than 47 thousand users and the number of downloads exceeds 6 million times.

Figure 2.29: OpenCV logo

2.4.2 Applications

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object detection
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical **machine learning** library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- K-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

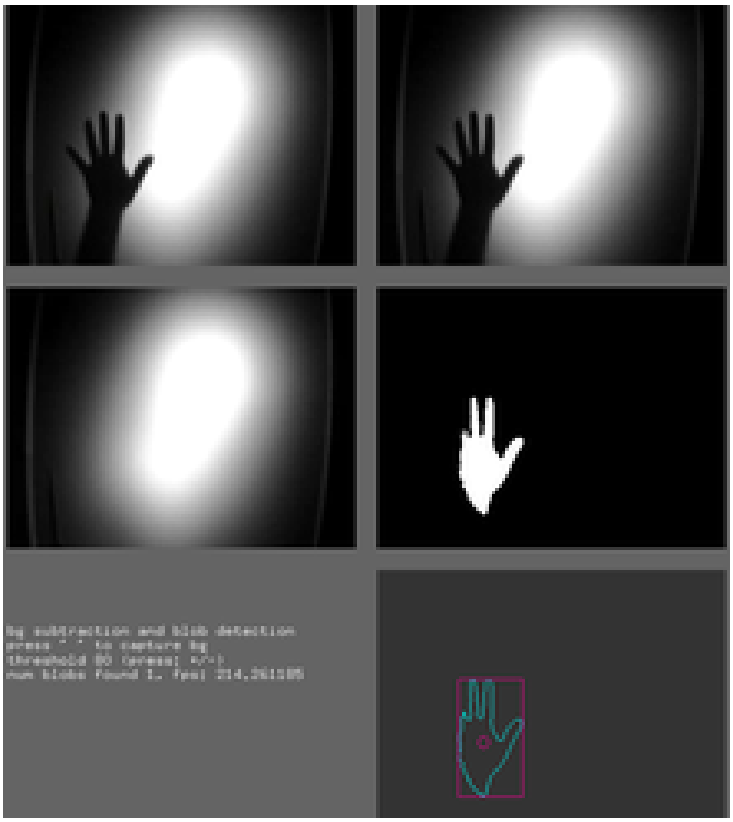

Figure 2.30: OpenFrameworks running the OpenCV add-on example

2.5 Python Programming Language

2.5.1 Introduce About Python Programming Language

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant

indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python is completely dynamically typed and uses automatic memory allocation, so it's similar to Perl, Ruby, Scheme, SmallTalk, and TCL. Python is developed in an open-source project, managed by the non-profit Python Software Foundation.

Originally, Python was developed to run on the Unix platform. But then over time, Python gradually expanded to all operating systems from MS DOS to Mac OS, OS/2, Windows, Linux and other operating systems of the Unix family.

2.5.2 Applications

Python is used in many different fields:

1. Web Development: Python provide many framworks to choose for development, Django framwork. Udacity, Youtube, Dropbox is built (in large part) by using Python.
2. Game Development: Pygame, but Python is not a best choice to to developed game.
3. Machine Learning: Theano, Tensorflow, Scikit-learn...
4. Computer Science: OpenCV, Numpy, Pandas, Scipy...
5. IoT: Arduino, Raspberry Pi

2.6 Introduction about Dlib

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge. In computer vision, Dlib has APIs help us do things like:

- Facial Landmark Detection: Facial Landmark is to identify locations like eyes, nose, mouth, face. It has many applications on mobile phone like (swap face, draw in face ...)
- Correlation Tracking
- Deep Metric Learning

Figure 2.31: Dlib and OpenCV library

Unlike OpenCV which provides algorithms for image processing and computer vision applications, dlib designed for machine learning and artificial intelligence applications with sub-libraries like:

1. **Classification**: Classification techniques are mainly based on 2 basic methods is kNN and SVM (Support Vector Machine)
2. **Data Transformation**: Quantitative transformation algorithms to reduce dimensionality, eliminate redundant data, and enhance distinctness of retained features
3. **Clustering**: Clustering techniques
4. **Regression**: Regression techniques
5. **Structure prediction**: Structure prediction algorithms
6. **Markov Random Fields**: Markov random fields algorithms

Specifically, the facial point detection is programmed using the dlib library with pre-provided training data and regression techniques.

2.7 Hardware Overview

In this project, the system includes the following hardware:

- **Raspberry Pi 3B+**: This is the center processing unit, which receives and processes the image signal received from camera and output it to speaker to warn drivers
- **Camera Pi**: Accquire image data to be processed
- **Speaker**: Warn to drivers
- **Mouse, keyboard**: Easier to manipulate and program on Raspberry.

2.8 Raspberry Pi 3B +

2.8.1 Introduction

Raspberry Pi is a word for computers with only one circuit board (also known as embedded computers) the size of a credit card, first developed in 2012 in the UK by the Raspberry Pi Foundation (a non-profit organization). profit) and produced by OEMs: Sony, Qsida, Egoma with the original purpose of promoting the teaching of basic computer science in schools and developing countries.

The original Raspberry Pi was based on a system on a **(SoC) BCM2835** chip of Broadcom, includes a **ARM1176JZF-S 700 MHz** processor. VideoCore IV GPU, originally shipped with 256MB RAM, after that is upgraded (B model and B+ model) upto 512MB. This board also have **Socket Secure Digital (SSD)** (model A and B) or microSD (model A+ and B+) use for boot device and presistent storage.

In 2014, Raspberry Foundation have published Compute module, pack a BCM2835 with 512MB RAM and an eMMC chip flash into a module for use as part of an embedded system. This organization provides Debian and Arch Linux ARM for users to download it. Tools available for Python as primary programming language, support for BBC Basic (through RISC OS image or Brandy Basic clone for Linux), C, C++, Java, Perl and Ruby.

This new board is the first to come in only one configuration (model B) and features a Broadcom BCM2836 SoC, with a quad-core ARM Cortex-A7 CPU and a dual-core VideoCore IV GPU; 1 GB of RAM with the remaining specifications similar to those of previous generations of B+ models.

29th January 2016, Raspberry Pi Foundation officially launched Raspberry Pi 3 with a lot of new improvements, especially on-board Wifi and Bluetooth support, Raspberry Pi 3 with ARM Cortex-A53 Quadcore 1.2GHz 64-bit CPU, 1GB RAM and especially supports 802.11n Wifi standard with Bluetooth 4.1. Besides, Raspberry Pi 3 is fully compatible with Raspberry Pi 2.

Raspberry Pi 3 Model B+ is an upgraded version of the previously released Raspberry Pi 3 Model B.

June 2019, Raspberry pi 4 launched with an upgrade over the previous generation with Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC ©1.5 GHz, there are 3 RAM options: 1GB, 2gb Or 4GB LPDDR4-2400 SDRAM, Gigabit Ethernet Network Port, 2 USB 3.0 and 2 USB 2.0 ports, and many other improvements.

2.8.2 Raspberry Pi 3B+

From 2012 to now, there have been many Raspberry Pi lines born with significantly improved functions and parameters. This Raspberry Pi 3 model B+ meets the more stringent requirements of users.

Some of the major improvements on the Raspberry Pi 3 Model B+ include the processor and networking capabilities. Model B + uses Broadcom BCM2837B0 quad-core 1.4GHz processor, clocked higher than BCM2837 1.2GHz on Pi 3 Model B.

The Model B+ offers wireless connectivity Wi-Fi in the 2.4GHz and 5GHz bands, gigabit Ethernet over USB 2.0 with speeds up to 300Mbps, 3 times faster than the Pi 3 Model B. The device supports Bluetooth 4.2 and Bluetooth Low Energy. Better connection with other smart devices.

Model B+ supports Power over Ethernet (PoE), which was not officially supported before. Raspberry Pi 3 upgrades processor and faster networking on Model B + but retains its compact size unchanged compared to previous versions.

Detail configuration of Raspberry Pi model 3B+ includes:

- **SoC chip**: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC ©1.4 GHz

- **RAM**: 1 GB LPDDR2 SDRAM
- **Wifi**: b/g/n/ac
- **Bluetooth**: 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- **40-pin GPIO (General Purpose Input Output)**: These pins are used to output control signals for leds and devices ... read the signal from sensors, switches ... There are also have I/O data transmission standards UART, I2C, SPI
- **4 x USB 2.0**: Connect to devices: mouse, keyboard, usb ...
- **MicroSD card slot**: Connect card to run OS
- **PoE**: Connect to Internet, VNC ...
- Improved PXE network and USB mass-storage booting
- **Power's jack**: MicroUSB, 5V power minimum 1A
- Better heat dissipation than Model B

There are some common advantages of Raspberry:

- Currently, Raspberry Pi is quite cheap with extremely compact size
- Very low energy consumption
- Powerful GPU
- This device can serve many purpose
- Raspberry Pi has ability to work continuously without stopping.

There are some common defects of Raspberry:

- Low CPU, can't handle heavy tasks
- To be able to use this device, users must have basic knowledge of Linux, electronics

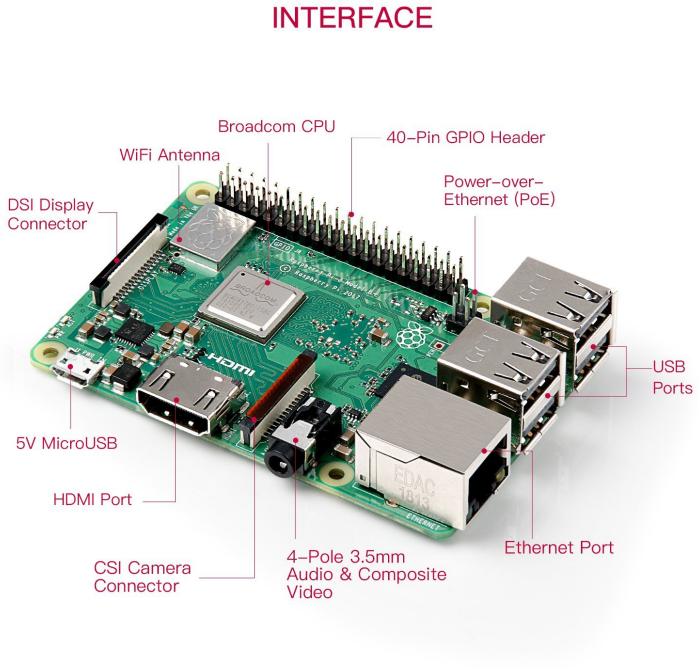

Figure 2.32: Raspberry Pi 3B+

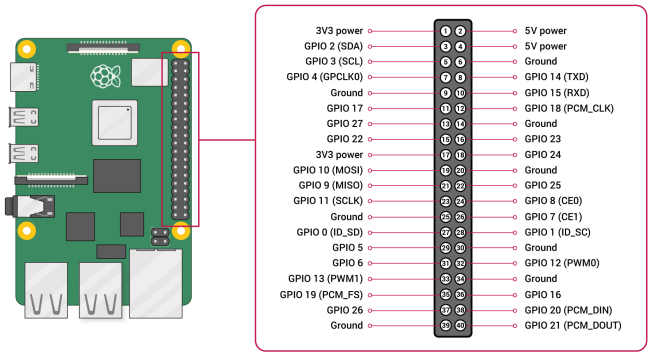

Figure 2.33: Pinout of Raspberry Pi 3B+

2.8.3 Operation System for Raspberry

Raspberry Pi has a lot of supported operating systems, including Raspbian which is the official operating system of the Raspberry Pi Foundation, in addition to 7 other operating systems that are confirmed to support and quite a few operating systems by developers. self-optimization. Some popular operating systems for Raspberry Pi such as: Raspbian, Ubuntu Mate, Snappy Core, Ubuntu, Windows 10 IoT Core, OSMC, OpenELEC, PiNet, RiscOS,...

1. **Raspbian**: This is the most popular, basic operating system and is provided by the Raspberry Pi Foundation itself. It is also recommended by the company, especially for beginners to get acquainted with RPI. Raspbian has a capacity after decompression is about 4GB, However, it is recommended to use a card of at least 8GB because free space is needed to install additional applications other uses.

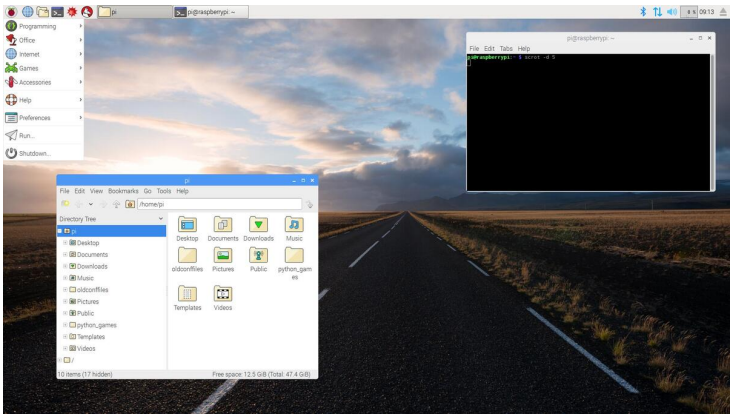

Figure 2.34: Raspbian OS desktop

2. **Ubuntu Mate**: Similar to Raspbian, Ubuntu Mate is also aimed at users who use Raspberry Pi as an office computer. However, Ubuntu Mate has a much nicer interface than Raspbian. It is very well optimized with Raspberry Pi 2 and 3, but to ensure the highest speed, Use a MicroSD card of class 6 or higher.

Figure 2.35: Ubuntu-Mate OS desktop

3. **Snappy Core Ubuntu**: It was created with the mission of running cloud applications and becoming an important part of IoT (Internet of Things), helping devices (phones, televisions, lights, fans, watches, rice cookers, ...) in life are perfectly connected. Its strength is that it can run many applications without a full Ubuntu operating system.

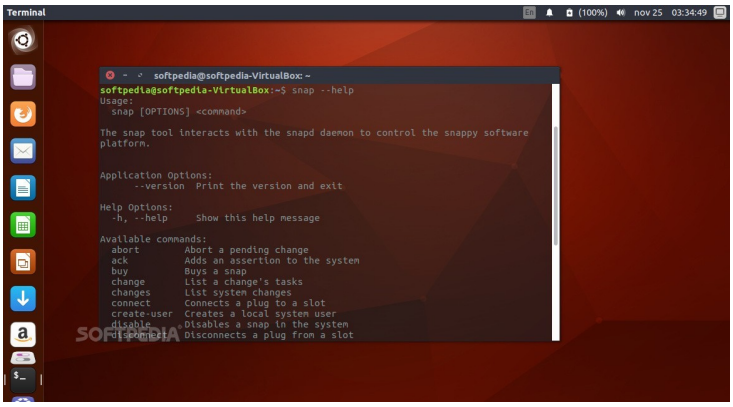

Figure 2.36: Snappy Core Ubuntu OS desktop

2.9 Camera Pi

The Raspberry Pi Camera v2 is a high quality 8 megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. ... In terms of still images, the camera is capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video. And there is some specifications:

- Weight: 3g
- Still resolution: 8 Megapixels
- Video modes: 1080p30, 720p60 and 640 × 480p60/90
- Linux integration: V4L2 driver available
- C programming API: OpenMAX IL and others available
- Sensor: Sony IMX219
- Sensor resolution: 3280 × 2464 pixels
- Sensor image area: 3.68 x 2.76 mm (4.6 mm diagonal)
- Pixel size: 1.12μm x 1.12μm
- Optical size: 1/4"
- Focal length: 3.04 mm
- Horizontal field of view: 62.6 degrees
- Vertical field of view: 48.8 degrees
- Focal ratio (F-Stop): 2.0

Chapter 3

INFERENCE

3.1 Block Diagram

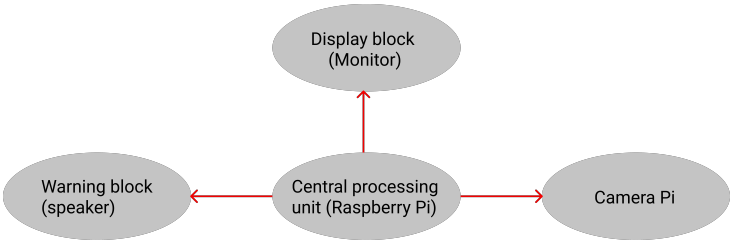

Figure 3.1: Block diagram of system

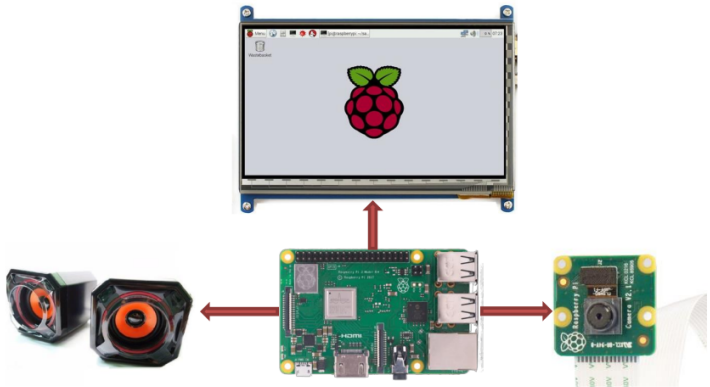

Figure 3.2: Device diagram of system

3.2 Flowchart

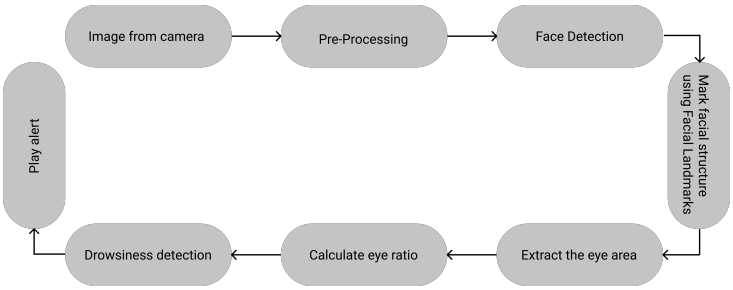

Figure 3.3: Flowchart of system

At first, we will set up a face-tracking camera to detect whether the driver is asleep or not (we are only interested in the eye area). Extract eye area and calculate eye ratio to determine if eyes are closed or open, when it is determined that the eyes are closed, we begin to count the number of frames that the eyes are closed, if the number of frames in which the eyes are closed exceeds the eye threshold (parameters specified by the programmer), the person is considered to have dozed off and starts to issue an alarm.

3.2.1 Image From Camera

Initially, we will establish a connection with camera pi and take each image frame for processing, the input image will be an RGB image. To access the ip camera, it is necessary to use the imutils library, which makes it easier to process images and work on OpenCV.

Figure 3.4: Input image

3.2.2 Pre-Processing

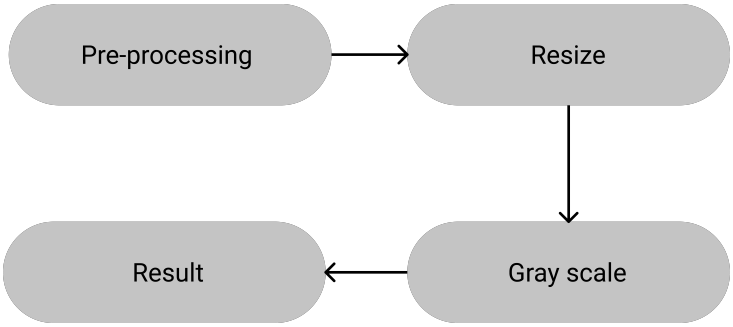

Figure 3.5: Pre-processing

Start capturing images continously with infinite loops, then do pre-processing by resizing to get the desired width (450 pixels) and convert it into gray-scale: "$gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)$"

Figure 3.6: Image after pre-processing

3.2.3 Face Detection Using Haar-Like (Haar Cascade)

Figure 3.7: Flowchart of face detection

To detect faces in an image, we need to convert the image to gray and then look at each single pixel and the points around it. Haar-Like features are black and white rectangles segmented into different regions. It's basically using Haar type features and then using as many of those features over many turns (cascade) to form a complete set of identifiers, and then proceed to draw recognizable faces on the corner photo.

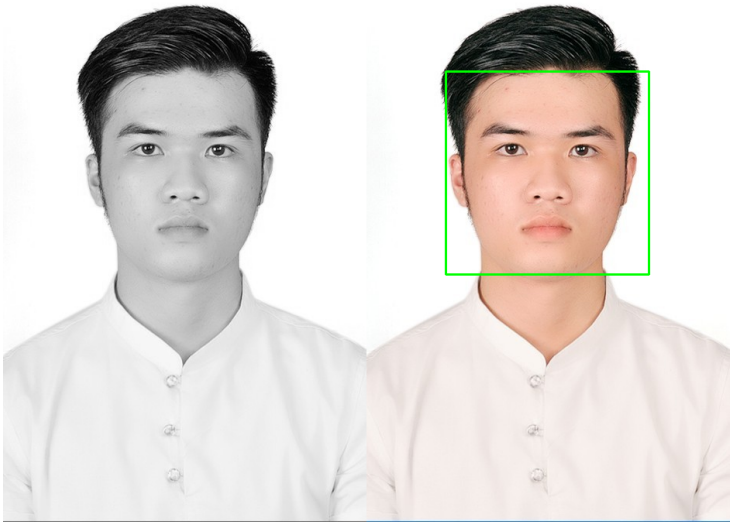

Figure 3.8: Face Detection

3.2.4 Mark Facial Structure Using Facial Landmarks

Figure 3.9: Facial landmarks

Using Dlib library's 68-point structure marking algorithm to locate facial parts including: eyes, nose, lip, eyebrows and face contour and then mark all these points on image.

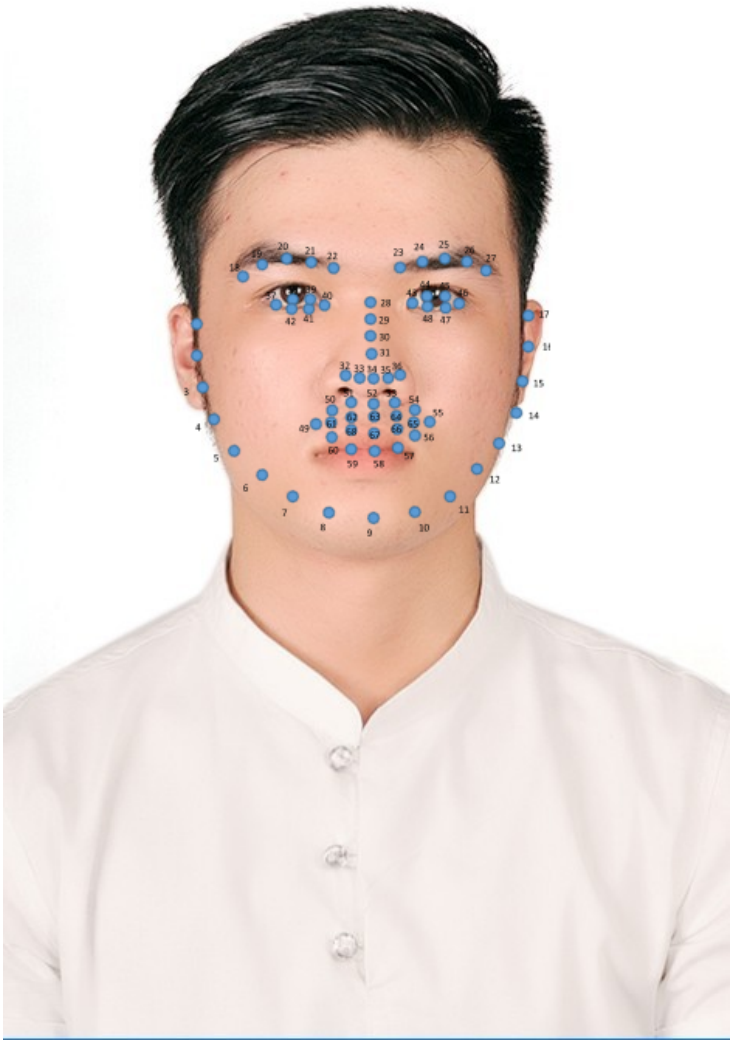

Figure 3.10: Mark 68 face points

3.2.5 Extract The Eye Area

In this project, we just care about the eye area so we extract the eye area by using cut array Numpy, we can extract the coordinate (x,y) of left eye and right eye.

3.2.6 Calculate Eye Ratio

With the extracted coordinate we will calculate the eye ratio, each eye is represented by 6 coordinates from P_1 to P_6.

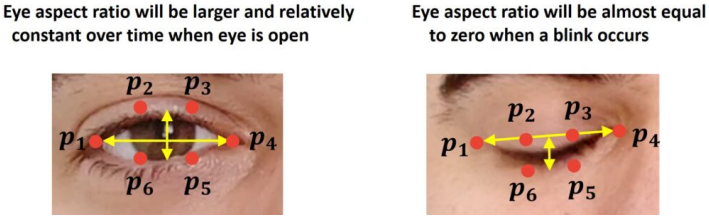

Figure 3.11: Extract eye ratio

There is a relationship between the width and height of these coordinates, we can use an equation that reflects this relationship called eye ratio (EAR).

$$EAR = \frac{||P_2 - P_6|| + ||P_3 - P_5||}{2||P_1 - P_4||} \tag{3.1}$$

Which $P_1 > P_6$ are landmarks marked on the eyes, the numerator of this equation calculates the distance between the vertical landmarks while the denominator calculates the distance between the horizontal landmarks.

The eye ratio is the distance that stays the same when the eyes are open, but rapidly decreases to zero when the eyes are closed or blinking.

By using this equation one can avoid complicated image processing techniques and simply rely on distance and time to determine whether the eyes are closed or open. To make this more clear, consider the picture below from Soukupová and Čech:

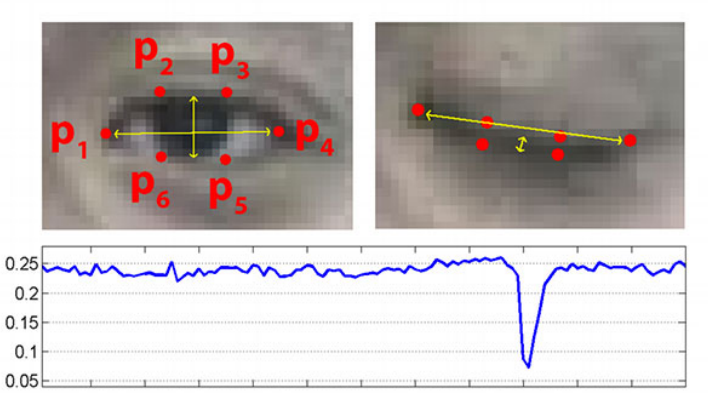

Figure 3.12: Top-left: A visualization of eye landmarks when then the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time.

3.2.7 Drowsiness Detection

To detect sleep need to first set the program parameters:

- Eye ratio threshold: to identify closed or open eyes. Determine the eye avg ratio by preparing 100 sample images including 50 eyes open and 50 closed eyes, examine each image to determine the threshold eye avg ratio, then average the eye avg ratio index to select the threshold of eyes open and closed, based on information This number is used to compare and determine whether the eyes are closed or open.
- Max sleep frames: to see if the person is awake or asleep
- $Alarmed = False$: initial state alarm is off

3.2.8 Alert

After determined that driver is asleep, we will turn on "$alarmed = True$" to play a loudspeaker warning to wake the driver (using playsound library).

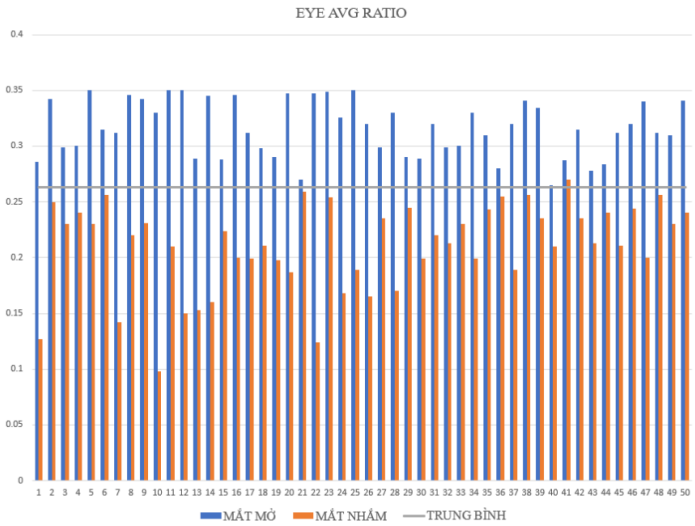

Figure 3.13: Eye average ratio chart with 100 eye samples

Chapter 4

EXPERIMENTAL RESULT

4.1 Realistic Model

Figure 4.1: Drowsiness detection model

4.2 Result

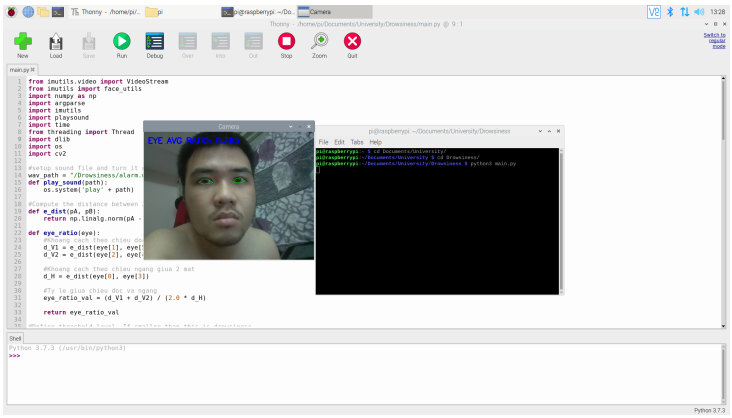

Figure 4.2: Eye aspect ratio

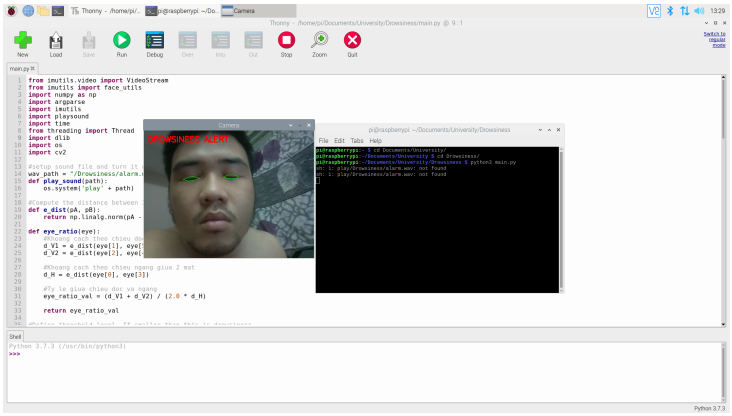

Figure 4.3: Drowsiness detected

Chapter 5

CONCLUSION AND TOPIC ORIENTATION

5.1 Conclusion

During the process of implementing this topic, I have built a system with practical applications for drivers when participating in traffic, which is a sleep detection system, built on the Linux and Linux operating system. Raspberry Pi 3B+ kit,with the support of OpenCV and Dlib libraries and programmed in Python programming language with the task of issuing warnings when drivers doze off.

Prodcut review:
To evaluate and comment objectively, I have tried to detect drowsiness in many different cases such as: front angle, tilt angle, bright enough, low light, the results obtained are shown in the version below. Based on the above table, we can see that the identification

Cases	Brightness enough	Lack of brightness
Front corner	91/100 images	53/100 images
Left tilt angle	82/100 images	Almost undetectable
Right tilt angle	76/100 images	Almost undetectable

Table 5.1: Results of eye state recognition in many cases

results with relatively high accuracy in the case of the frontal face in bright enough conditions, for the left and right tilt angles, the recognition process will be more difficult. a bit, but the camera still catches the eye with high accuracy (in bright enough conditions)

On the contrary for the above three cases, the accuracy will gradually decrease in low light and almost undetectable for the case of right tilt angle (low light conditions) and tilt angle greater than 40 degrees (both bright enough and low light).

5.2 Limitations

5.2.1 Advantages

- Compact, convenient due to the small size of the Raspberry pi, it can be integrated directly on the car without taking up too much space, can be integrated based on the existing security systems on the vehicle

- As a programmed application, it can save costs, the design and construction is also quite simple because the hardware is simple, does not require complicated components
- Relatively high accuracy
- The program meets the actual requirements, the image processing time is fast, and the warning is issued in real time

5.2.2 Disadvantages

- Depends heavily on external conditions such as light, rotation angle,...
- Face to face directly, if turning left or right with an angle greater than 40 degrees or the end of the head, raising the head more than 30 degrees, it may not be detected

5.3 Development Orientation

- Research using new, optimized and better recognition algorithms
- For the convenience of use, it is necessary to research and develop application software that can be controlled via smartphones
- Developing more diverse warning systems such as vibrating seats, or calling directly to the driver's phone, ... to wake up more effectively.
- When the driver shows signs of falling asleep, we can activate the car's self-driving mode and send a continuous warning to wake the driver.

Chapter 6

APPENDICES

6.1 Drowsiness Detection Program

```
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import argparse
import imutils
import playsound
import time
from threading import Thread
import dlib
import os
import cv2

```

Listing 6.1: Import library

```
wav_path = "/Drowsiness/alarm.wav"
def play_sound(path):
    os.system('play' + path)

```

Listing 6.2: Setup sound file and turn it on

```
def e_dist(pA, pB):
    return np.linalg.norm(pA - pB)
def eye_ratio(eye):
    #Khoang cach theo chieu doc mi tren va mi duoi
    d_V1 = e_dist(eye[1], eye[5])
    d_V2 = e_dist(eye[2], eye[4])

    #Khoang cach theo chieu ngang giua 2 mat
    d_H = e_dist(eye[0], eye[3])

    #Ty le giua chieu doc va ngang
    eye_ratio_val = (d_V1 + d_V2) / (2.0 * d_H)

    return eye_ratio_val

```

Listing 6.3: Compute the distance between 2 point A and B

```
eye_ratio_threshold = 0.25

```

Listing 6.4: Define threshold level. If smaller than this is drowsiness

```
max_sleep_frames = 16

```

Listing 6.5: Threshold so frame lien tuc nham mat

```
        sleep_frames = 0

```

Listing 6.6: Dem so frame ngu

```
        alarmed = False

```

Listing 6.7: Check xem da canh bao hay chua

```
        face_detect = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
        landmark_detect = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

```

Listing 6.8: Khoi tao cac module detect mat va facial landmark

```
        (left_eye_start, left_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
        (right_eye_start, right_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

```

Listing 6.9: Lay danh sach cac cum diem landmark cho 2 mat

```
        vs = VideoStream(src=0).start()
        time.sleep(1.0)

        while True:

            # Doc tu camera
            frame = vs.read()

            # Resize de tang toc do xu ly
            frame = imutils.resize(frame, width=450)

            # Chuyen ve gray
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Detect cac mat trong anh
            faces = face_detect.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
    minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)

            # Duyet qua cac mat
            for (x, y, w, h) in faces:

                # Tao mot hinh chu nhat quanh khuon mat
                rect = dlib.rectangle(int(x), int(y), int(x + w),
                    int(y + h))

                # Nhan dien cac diem landmark
                landmark = landmark_detect(gray, rect)
                landmark = face_utils.shape_to_np(landmark)

                # Tinh toan ty le mat phai va trai va trung binh cong 2 ratio
                leftEye = landmark[left_eye_start:left_eye_end]
                rightEye = landmark[right_eye_start:right_eye_end]

                left_eye_ratio = eye_ratio(leftEye)
                right_eye_ratio = eye_ratio(rightEye)

                eye_avg_ratio = (left_eye_ratio + right_eye_ratio) / 2.0

                # Ve duong bao quanh mat
                left_eye_bound = cv2.convexHull(leftEye)
                right_eye_bound = cv2.convexHull(rightEye)
                cv2.drawContours(frame, [left_eye_bound], -1, (0, 255, 0), 1)
                cv2.drawContours(frame, [right_eye_bound], -1, (0, 255, 0), 1)

                # Check xem mat co nham khong
                if eye_avg_ratio < eye_ratio_threshold:
                    sleep_frames += 1
                    # if the eyes were closed for a sufficient number of
                    # frames, then sound the alarm
```

```
                    if sleep_frames >= max_sleep_frames:

                        if not alarmed:
                            alarmed = True
                            # Duong dan den file wav


                            # Tien hanh phat am thanh trong 1 luong rieng
                            t = Thread(target=play_sound,
                                       args=(wav_path,))
                            t.deamon = True
                            t.start()

                        # Ve dong chu canh bao
                        cv2.putText(frame, "DROWSINESS ALERT", (10, 30), cv2.
FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

                # Neu khong nham mat thi
                else:

                    # Reset lai cac tham so
                    sleep_frames = 0
                    alarmed = False

                    # Hien thi gia tri eye ratio trung binh
                    cv2.putText(frame, "EYE AVG RATIO: {:.3f}".format(eye_avg_ratio),
(10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2)

            # Hien thi len man hinh
            cv2.imshow("Camera", frame)

            # Bam Esc de thoat
            key = cv2.waitKey(1) & 0xFF
            if key == 27:
                break

```

Listing 6.10: Doc tu camera

```
        cv2.destroyAllWindows()
        vs.stop()

```

Listing 6.11: End program

6.2 Install Raspbian OS for Raspberry Pi 3B+

Require equipment for installation:

- Micro SDHC card 32GB + adapter
- Laptop for installation
- Monitor
- LAN cable for convenient remote control by laptop
- Power 5v, 2.5A (at least 1A)

6.2.1 Step 1

Download Raspbian OS ISO file to laptop and extract it

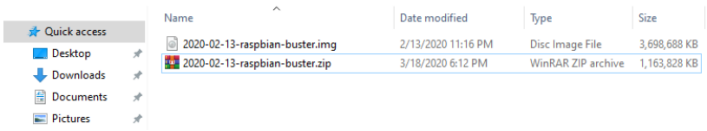

Figure 6.1: The file contain Raspbian OS

6.2.2 Step 2

Install SD card formater and win32 Disk Imager software

6.2.3 Step 3

Insert MicroSD card into card reader in laptop and check drive name of disk assigned to memory card to avoid confusion causing other data loss.

6.2.4 Step 4

Run SD Card Formater software to format memory card then run Win32 Disk Imager start install OS to the memory card

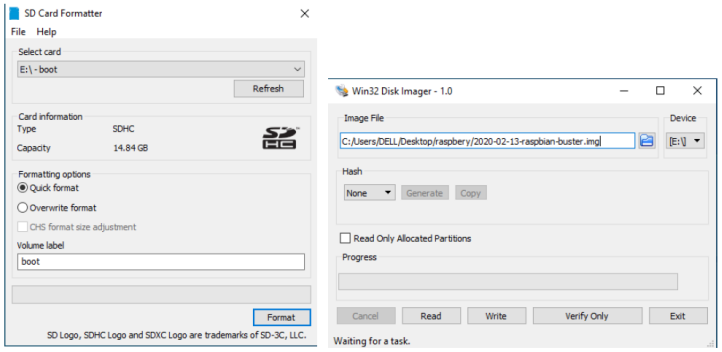

Figure 6.2: Format disk and install OS into memory card

6.3 Install Necessary Libraries

6.3.1 OpenCV

$sudo apt install python3-pip

This step allow us to install pip in Python 3. Next we using Pip command to install OpenCV library by using:

$sudo pip3 install opencv-python

Next we install another package to run the code:

- imutils
- numpy
- playsound
- dlib

6.4 Run program

At last, we run this command to run program with 2 components is sound file and data file of HaarCascade front face:

$python3 main.py

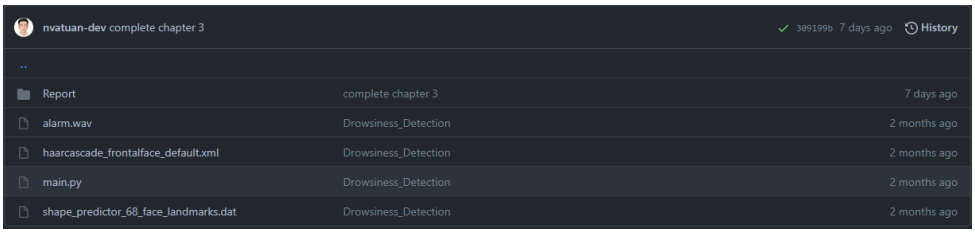

Figure 6.3: Folder tree of project

THE REFERENCES

1. Adrian Rosebrock, May 8th 2017, "*Drowsiness detection with OpenCV*"
2. Nguyen Chien Thang, February 26th 2017, "*Computer Vision & Pi – Lap dat Pi tren xe hoi de phat hien tai xe ngu gat*"
3. Raspberry Pi, January 28th 2019, "*Raspberry Pi Compute Module 3+*"
4. Hai Ha, February 20th 2019, "*Histogram of Oriented Gradients*"
5. Adrian Rosebrock, April 3rd 2017, "*Facial landmarks with dlib, OpenCV, and Python*"
6. Adrian Rosebrock, April 24th 2017, "*Eye blink detection with OpenCV, Python, and dlib*"
7. Wikipeidia, April 13th 2021, "*Euclidean distance*"
8. OpenCV, May 29th 2021, "*Cascade Classifier*"