Software & Information Systems Engineering Department

Smart Person Identification System for Home/Office Visitors

SE/ISE Capstone Project Phase 2

Data 18/01/2021

Authors:
Carmela Rotzaid

Supervisor:
Dr. Avi Soffer

Table of Contents

1. ABSTRACT

Privacy and Security are two universal rights and, to ensure that in our daily lives we are secure, a lot of research is going on in the field of home and work place (like office) security, where we connect objects to share data. People always try to make life easier at the same time. In this project I use data extraction methods to find personal features from the digital human face image.
The algorithm has main iterations: capture, preprocessing, comparison and finally classification.
The identification process is done by Deep neural network (DNN) using the "deep metric learning" method. The face detection classification Is done by re-trained wide-Haar cascade and finally, the face recognition is based on resnet-34.

Keyword -Raspberry Pi, Facial Detection, Facial Recognition, Home Security, office Security.

2. INTRODUCTION

"Smart Person Identification System for Home / Office Visitors" is a home and office security product. The world of connectivity and smart devices there is an urgent need to modify our existing day to day objects and make them smart, also it is not the era when we can blindly trust the old and conventional security measures, specifically speaking is our door locks. To change and modernize any object we need to add extra functionality. The major drawbacks in a common door lock are that anyone can open a conventional door lock by duplicating or stealing the key and its simply impossible if we want our friends and family to enter our house, without being actually present over there. So, to simply convert this normal door lock into a smart lock, which can open the door whenever we turn up in front of the gate or when we want it to open up for someone else without being physically present, after we receive mail with arrival details and face photo attached. So, an era has come where devices can interact with its users and at the same time ensure of their safety and keep improvising themselves. Users could operate on a touchscreen to select entering the house by recognizing the face. For face recognition, an image is captured by the pi camera and pre-processed by Raspberry pi like converting, re-sizing, and cropping. Then face detection and recognition are performed. Once the face is recognized by the classifier based on a pre-stored image library, the image will be compared to images stored in the Data base, but if the identification process fails, the image will be e-mailed to the owner. The system based on Raspberry-pi 4 and algorithms from OpenCV and Dib libraries. Also, sending Email, is done by raspberry-pi using SMTP (Simple Mail Transfer Protocol) for sending and receiving emails. And the door is opened by general-purpose input / output (GPIO).

3. Background and Related Work

3.1. k-means clustering

k-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriority. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed, and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

3.2. Support vector machine

A Support Vector Machine (SVM) is a supervised learning technique that constructs a hyperplane or a set of hyperplanes in a high-dimensional space by best separating the training examples according to its assigned class.
This can be seen in the next diagram, where the green line is the representation of the hyperplane that best separates the two classes because the distance to the nearest element of each of the two classes is the largest:

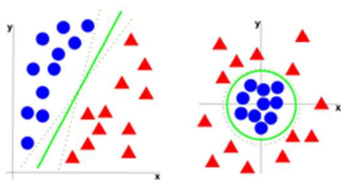


Fig 1

In the first case, the decision boundary is a line while, in the second case, the decision boundary is a circumference. The dashed lines and the dashed circumferences represent other decision boundaries, but they do not best separate both classes.
The objective of the Support Vector Machine is to distinctly classify data points, in an N-dimensional space, when N is the number of features, using a hyperplane for separation.

3.3. Histogram of Oriented Gradients (HOG)

HOG is a feature descriptor that uses for object detection. HOG represents objects as a single feature vector as opposed to a set of feature vectors where each represents a segment of the image. The idea in HOG is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The first step is to divide the image into small cells and a histogram of edge orientations is computed for each cell. The second step is to discretize each cell into angular bins according to the gradient orientation, then each cell's pixel contributes weighted gradient to its corresponding angular bin. Groups of adjacent cells are considered spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor. Demonstration of the HOG algorithm implementation scheme:

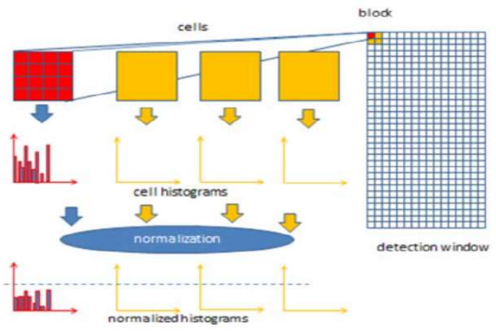


Fig2. Demonstration of the HOG algorithm implementation scheme.

3.4. Residual Networks (ResNet)

Residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do that by utilizing skip connections, or short-cuts to jump over some layers. Typical ResNet models are implemented with double-layer (or triple-layer) skips that contain nonlinearities (ReLu) and batch normalization in between as shown in Figure. ResNet-50 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 50 layers deep and can classify images into 1000 object categories, such as keyboard, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images and can uses for face features classification. The network has an image input size of 224-by-224.

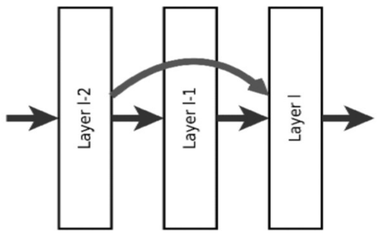


Fig3. Residual network layer-skip example.

3.5. Haar Cascade classifier

A Haar Cascade is basically a classifier which is used to detect the object for which it has been trained for, from the source.

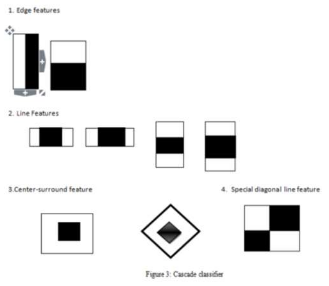

1. Edge features
2. Line Features
3.Center-surround feature
4. Special diagonal line feature
Figure 3: Cascade classifier

Figure 4

This proposed system uses Haar Cascades classifier as a face detection algorithm. Firstly, the algorithm needs a lot of positive images and negative images to train the Haar cascades classifier. Positive images are images with clear faces where negative images are those without any faces. Haar cascades are similar to the convolutional kernel which are shown below in figure 3.

$$I_{\Sigma}(x,y) = \sum_{x' \leq x} i(x' + y')$$

Each feature is represented as a single value obtained from the difference of the sums of pixels in white rectangle from the sum of all pixels in the black rectangle. All different possible sizes and locations of classifier is used for calculating of plenty of features. As the number of classifiers increase the arithmetic computations seems to take a long time. Instead of it, the concept of Integral Image has been used. Image Processing Integral image is a data structure which is a summed area table and algorithm for quickly and efficiently generating sum of values in a rectangular grid subset. Integral image is derived by using following equation, to avoid the complexity of calculation we use Adaboost machine learning algorithm, which is inbuilt in OpenCV library that is cascade classifier, to eliminate the redundancy of the classifiers. Any classifier which has a probability of 50% of more in detection is treated as weak classifier. The Sum of all weak classifier gives a strong classifier which makes the decision about detection. Although it is very vague to classify with one strong classifier, we use the cascade of classifiers. Classification takes place in stages, if the selected region fails in the first stage, we discard it. We don’t use the classifiers on that region which is discarded. The region which passes all the stages i.e. all strong classifiers is treated as the detected face. Detected Faces are passed to the Face recognition phase. Local Binary Patterns histogram algorithm (LBPH) has been used for face recognition. Local binary patterns are simple at the same time very efficient texture operator which assigns the pixels of the image by comparing with the adjacent pixels as threshold and which results in a binary result. The detected integral image is subjected to this Local binary pattern. Face recognition is extremely vulnerable to the environment changes like brightness, facial expressions, and position. Face pre-processing is the module which reduces the problems that makes the picture unclear to recognize the face such as less brightness and contrast problems and noise in the image and make sure the facial features always be in a constant position. In this project I use Haar Cascade Classifier and Histogram of Oriented Gradients (HOG) image descriptor and a Linear Support Vector Machine (SVM) for face pre-processing.

3.6. Adaboost

Adaboost is a technique that uses multiple classification algorithms throughout the training data. First, we take a random sample from the dataset and call it a *training set*. Adaboost assigns weights to every example in the training set. After that, the first classifier is run. When the classification is done, Adaboost looks at the misclassified examples in the training set. They are then assigned greater weights, as are the correctly classified examples. This happens so that when the second classifier is run, it takes into consideration examples with greater weights. In addition, weights are assigned to classifiers as well. When the classifier gives an output, its error is calculated. Based on the intensity of the error, weights are assigned. If the error is less than 50% accurate, the classifier is given a negative weight; otherwise, it's given a positive weight. Fifty percent accuracy equates to a weight of zero.

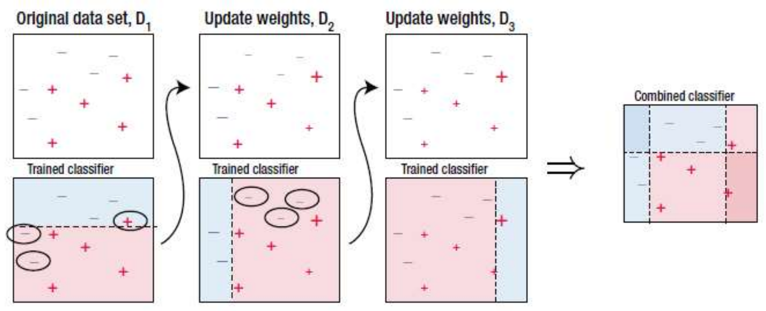


Figure A-1. *(Credit: Research Gate)*

Adaboost is used most widely in face recognition. We have already seen how we used Haar cascades, or LBPH cascades, for detecting the faces. Adaboost plays a very big role in that.

3.7. Simple Mail Transfer Protocol (SMTP)

SMTP is the standard protocol for providing email services on a TCP/IP network. This server provides the ability to receive and send email messages.
SMTP is an application-layer protocol that provides the delivery and transmission of email over the Internet. It is maintained by the Internet Engineering Task Force (IETF). SMTP is generally summed within an email client application and is composed of four key components:

- ❖ Local user or client-end utility known as the mail user agent (MUA)
- ❖ Server known as mail submission agent (MSA)
- ❖ Mail transfer agent (MTA)
- ❖ Mail delivery agent (MDA)

Getting email alerts or set of data using raspberry pi python program is very useful application. All we need is SMTP library in the python script. There are many versions of python, but pi is more compatible version of it. Below are the mentioned steps of sending SMTP email using pi.

3.8. Servo Motor

There are many types of servo motors and their main feature is the ability to precisely control the position of their shaft. A servo motor is a closed-loop system that uses position feedback to control its motion and final position. In industrial type servo motors the position feedback sensor is usually a high precision encoder, while in the smaller RC or hobby servos the position sensor is usually a simple potentiometer. The actual position captured by these devices is fed back to the error detector where it is compared to the target position. Then according to the error the controller corrects the actual position of the motor to match with the target position.

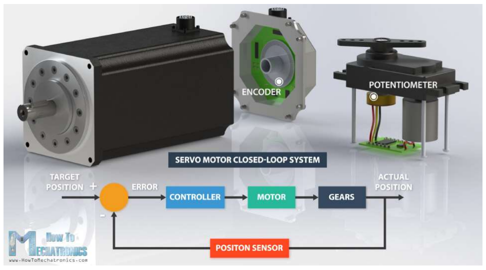
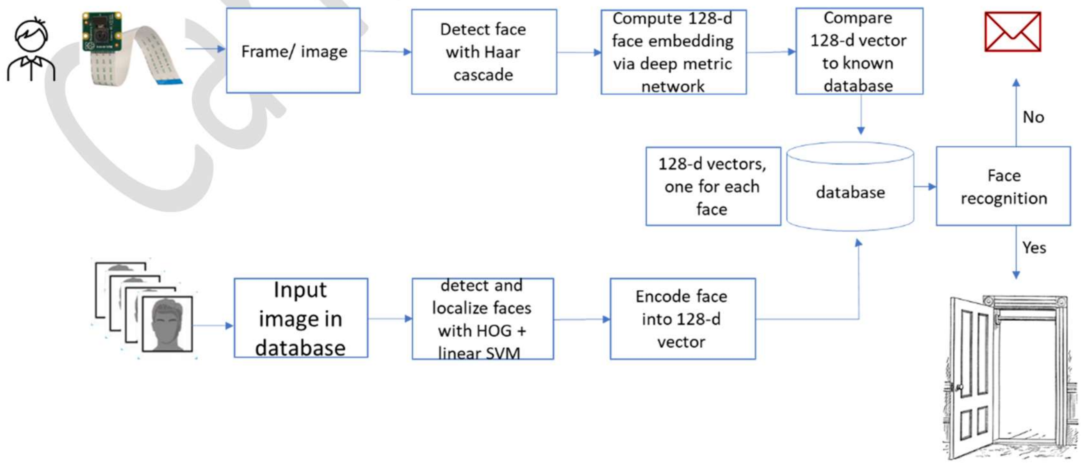


Fig5

4. Project Achievements

4.1. System design

Many face detection and face recognition algorithms can be used.
The algorithms Suggested In this section I explain how to apply face detection using the Haar cascade, HOG and Linear SVM.
For face recognition, the system will use of Face Recognition Library for Python, OpenCV and Dlib library. The open source Dlib library, OpenCV library and Face Recognition library for Python contain built-in face recognition algorithms That have been used in this system.
The system consists of three subsystems - dataset creation, dataset training, and testing.


Frame/ image
Detect face with Haar cascade
Compute 128-d face embedding via deep metric network
Compare 128-d vector to known database
No
128-d vectors, one for each face
database
Face recognition
Yes
Input image in database
detect and localize faces with HOG + linear SVM
Encode face into 128-d vector

4.2. Dataset Creation

First, we create of a folder containing images of a person.
The folder name should be the name of the individual whose photos are contained within.

4.3. Face Detection with HOG and linear SVM

The system will check if there is any human face in the given image. If there is not – the system will reject that image, else, classifier will identify the face areas and create a separate image for every face in the picture. The face detection is done by trained SVM with Histogram of Oriented Gradients (HOG) descriptors. These days SVM with HOG descriptors mainly use for face and shape recognition. We choose the SVM with HOG algorithm because of his accuracy and popularity. The SVM is trained with two groups of samples, P and N, where P is face images (positive samples) and N is non-face images (negative samples). Next, 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 HOG descriptors and apply the classifier. If the classifier incorrectly classifies a given window as face (false positives), record the feature vector

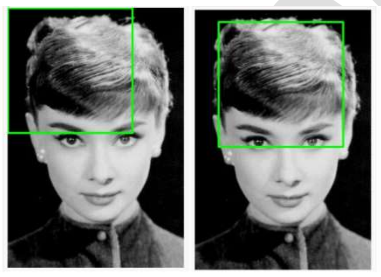


Figure 6.(a) Sliding window technique examples.

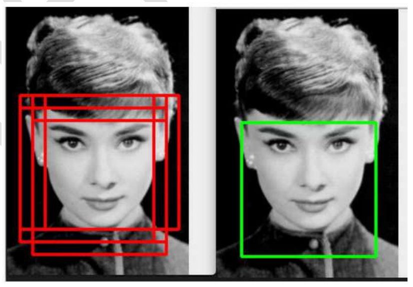


Figure 6.(b) Face detection using SVM with HOG descriptors.

4.3.1. Pre-processing

The face images are still not ready and need to be normalized. Every image is different from the other in many perspectives, for example, every image has different lighting conditions and different presence of noisy data. Therefore, the face images will be normalized by 3 steps.
The result of this phase is a set of normalized face images.

4.3.2. Resize detected face image

Each face image has a different height and width. Therefore, the first step is to standardize the face images sizes and each image will modify to standards height and width (Example 224 X 224).

4.3.3. Colour conversion

Each face image has different standards of colour format. Therefore, the second step is to convert the images formats to one standard format. The standard format that we decide to use is the grayscale format. This format helps us to get a uniform distribution of the intensity of values in the images. To get this conversion the algorithm needs to take every pixel in the image and convert his RGB values to grayscale value. The following formula is using to convert the RGB image into a grayscale image:

$$G(x, y) = 0.21R + 0.71G + 0.07B$$

4.3.4. Noise reduction

The final step for image normalization is noise reduction, it means that we need to reduce all the noises from our images. The cause for image noises could happen from dirt on camera lenses, an imperfection of the camera flashlight or from any other reason. The face images may contain some noises that we need to get rid of them, thus, all the face images are sent to noise reduction filter. Gaussian smoothing uses for a noise reduction filter. Gaussian smoothing of (x, y) can be given as:

$$G\sigma \equiv \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

To calculate the weight of each pixel in the image we can use this function. Assume that the point (0,0) is the center point of the weight matrix, then, the nearest coordinates of the center point can represent as:

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

example for weight matrix where σ = 1 is:

0.0453	0.0556	0.0453
0.0556	0.0707	0.0556
0.0453	0.0556	0.0453

The weighted matrix sum is calculated by the formula:

$$sum(w) = \sum_{i=1}^{9} w(i)\text{, where } i \text{ is the index of the pixel in the matrix}$$

Then the algorithm calculates the weighted average of the nine points in the matrix using the formula:

$$avg(w) = \frac{w(i)}{sum(w)}$$

To calculate the Gaussian blur for each pixel, the pixel colour value is multiplied by the average weighted value. The pixel colour value is between 0 – 255. The Gaussian blur calculate:

$$G(i) = pixelColorValueOf(i) * avg(w)$$

These values of the matrix help to calculate the Gaussian blur value for the centre point.

$$blur(middle) = \sum_{i=1}^{9} G(i)$$

The Gaussian blur of the image can be calculated by repeating the above steps for all the points in the image. An example of a Gaussian blur is shown in Figure 7.

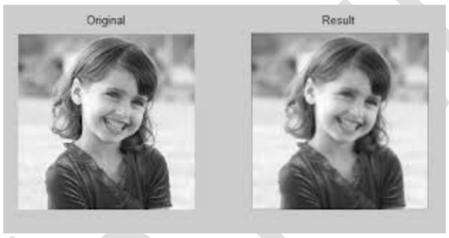


Figure 7. Example of Gaussian blur.

4.4. Detection with Haar cascade classifier

The system detects a human face in a video stream using Haar cascade classifier.
The Haar cascade algorithm as known use machine learning object detection, to identify objects in images and videos based on the concept of features.
The cascade function is trained on a huge dataset of negative and positive images. For face detection, the algorithm requires positive images that contain faces and negative Images that do not contain faces to train the classifier, which then is used to extract the features from. Every feature is a single value obtained by calculating the difference between the sums of pixel intensities in each region. Due to the quantity of features used, integral image concept is applied to prevent an increase in computation time. It also helps to simplify pixel calculations.

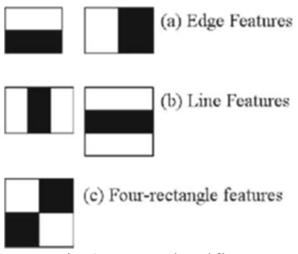


Fig 8: Haar classifier

4.5. Adaboost

Adaboost is used for feature selection, reducing the complexity of classifiers and training them. After which each feature is applied on the training images and for every feature, the best threshold is found to classify the faces as positive or negative. However, since it is inefficient and time consuming, cascading classifiers are used to obtain the best features from a face by grouping the features into a stage of classifiers and processing or discarding the face region according to the stages of features passed.

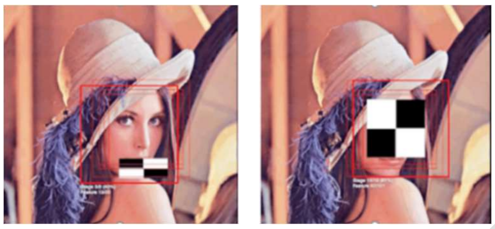
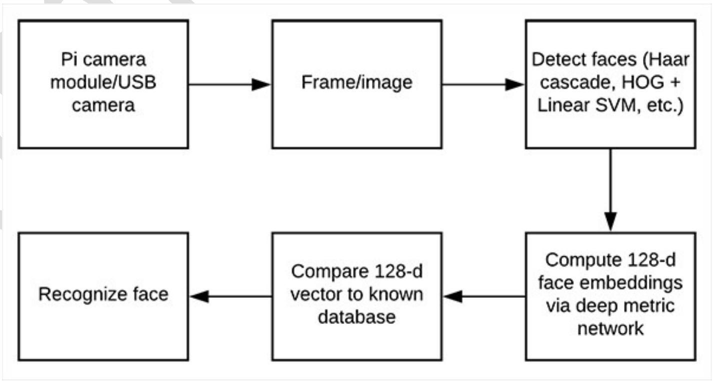


Fig 9

4.6. face recognition

Beginning with capturing input frames from our Raspberry Pi, our workflow consists of detecting faces, computing embeddings, and comparing the vector to the database via a voting method. OpenCV, dlib, and face recognition is required for this face recognition method.
Face recognition with the development of computer vision, machine learning, and deep learning. Face recognition can be applied to a wide range of uses, including crime prevention, surveillance, forensic applications, biometrics, and, more recently, in social networks. Automatic face recognition has various challenges, such as occlusions, appearance variations, expression, aging, and scale variations.


Pi camera module/USB camera
Frame/image
Detect faces (Haar cascade, HOG + Linear SVM, etc.)
Compute 128-d face embeddings via deep metric network
Compare 128-d vector to known database
Recognize face

4.7. facial embeddings with deep metric learning

The technique called deep metric learning. we typically train a network to:

- ❖ Accept a single input image
- ❖ And output a classification/label for that image

However, deep metric learning is different.
Instead, of trying to output a single label (or even the coordinates/bounding box of objects in an image), we are instead outputting a real-valued feature vector.
In this Project, I use of DLIB facial recognition network, the output feature vector is 128-dimensional (list of 128 real-valued numbers) that is used to quantify the face. Training the network is done using triplets:

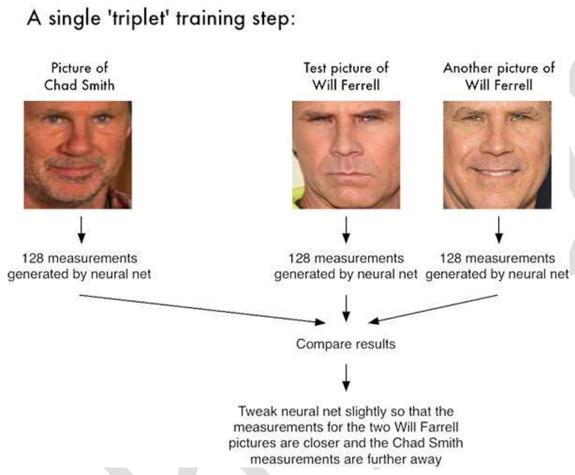
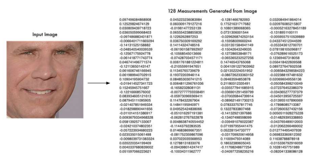

A single 'triplet' training step:
Picture of Chad Smith
Test picture of Will Ferrell
Another picture of Will Ferrell
128 measurements generated by neural net
128 measurements generated by neural net
128 measurements generated by neural net
Compare results
Tweak neural net slightly so that the measurements for the two Will Farrell pictures are closer and the Chad Smith measurements are further away

Figure 10: Facial recognition via deep metric learning involves a "triplet training step." The triplet consists of 3 unique face images — 2 of the 3 are the same person. The NN generates a 128-d vector for each of the 3 face images. For the 2 face images of the same person, we tweak the neural network weights to make the vector closer via distance metric.

Dlib offers a high-quality face recognition algorithm based on deep learning.
Dlib implements a face recognition algorithm that offers state-of-the-art accuracy.
More specifically, the model has an accuracy of 99.38% on the labelled faces in the wild database.
The implementation of this algorithm is based on the ResNet-34 network proposed in the paper Deep Residual Learning for Image Recognition (2016), which was trained using three million faces.
This network is trained in a way that generates a 128-dimensional (128D) descriptor, used to quantify the face. The training step is performed using triplets. A single triplet training example is composed of three images. Two of them correspond to the same person. The network generates the 128D descriptor for each of the images, slightly modifying the neural network weights in order, to make the two vectors that correspond to the same person closer and the feature vector from the other person further away. The triplet loss function formalizes this and tries to push the 128D descriptor of two images of the same person closer together, while pulling the 128D descriptor of two images of different people further apart.

4.8. Encoding face image

This process of training a convolutional neural network to output face embeddings requires a lot of data and computer power. Even with an expensive NVidia Telsa video card, it takes about 24 hours of continuous training to get good accuracy. But once the network has been trained, it can generate measurements for any face, even ones it has never seen before. So, this step only needs to be done once. So, all we need to do ourselves is run our face images through their pre-trained network to get the 128 measurements for each face. Fig11: Here is the measurements for our test image:



0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

4.9. Finding the person name from the encoding

This last step is the easiest step in the whole process. All we must do is find the person in our database of known people who has the closest measurements to our test image.
You can do that by using any basic machine learning classification algorithm. We will use a simple linear SVM classifier, but lots of classification algorithms could work.
All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds.
The result of the classifier is the name of the person.

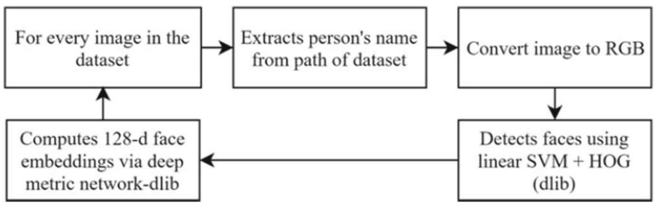
4.10. Training

Deep Neural Network basically consists of discovering the right values on each of the filters.
During the training process, the images are passed through the network by applying several filters on the images at each layer. The values of the filter matrices are multiplied with the activations of the image at each layer. The activations coming out of the final layer are used to find out which class the image belongs to. When a deep network is being trained, the main goal is to find the optimum values on each of these filter matrices so that when an image is propagated through the network, the output activations can be used to accurately find the class to which the image belongs.
Later, the error value is being calculated in order to make the network reliable by adjusting the weights in each layer until minimum error is achieved.

4.11. Dataset Training

HOG and Linear SVM algorithms are used for face detection.
For every image in the dataset, the person’s name is extracted from the path, the image is converted to RGB, faces are localized in the image using HOG and Linear SVM.

The Face Recognition library uses the Dlib module to compute face embeddings using
Dlib's deep metric network. A pickle (file) with 128-d embeddings for each face in the dataset along with their names is created.


For every image in the dataset
Extracts person's name from path of dataset
Convert image to RGB
Detects faces using linear SVM + HOG (dlib)
Computes 128-d face embeddings via deep metric network-dlib

4.12. Dataset Testing

The known 128-d face embeddings are loaded, and the video stream is initialized with
OpenCV's Haar cascade for localizing and detection of faces.
The Pi-camera module then starts video streaming to connected devices in real-time.
A frame is captured from the threaded video stream and resized for pre-processing.
OpenCV is then used for converting the input frame to grayscale for face detection and to RGB for face recognition. Faces in the grayscale frame are then detected using
Haar cascade and bounding box coordinates are returned.
Face Recognition library is then used to compute facial embeddings for each face in the bounding box. The Face Recognition library makes use of k-NN algorithm for calculating the
Euclidean distance between each the candidate facial embedding and the known facial embeddings of the dataset. This distance shows how similar the faces are.
If the calculated distance is above a tolerance of 0.6, then "True" is returned, indicating the faces match. Otherwise, false is returned. A list of True/False values, one for each image in the dataset is returned. The indexes of the images with "True" values are stored and the
name associated with each index is used to count the number of "votes" for each name.
The recognized face with the largest number of votes is selected. This is done for each detected face in the frame. The predicted name is then used to label the faces in the frame.
The resultant output is video streamed in real-time for other connected devices to view.

$$d(p,q) = d(q,p)$$
$$= \sqrt{(q_{1-}p_1)^2 + (q_3 - p_3)^2 + \cdots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i - p_i\ )^2}$$

Fig: Euclidean distance

4.13. Raspberry Pi Door Lock Circuit Diagram

The circuit diagram for Raspberry Pi Servo Door Lock is very simple as because we only need to connect the servo door lock to Raspberry Pi. Solenoid Lock needs 9V-12V to operate and Raspberry Pi GPIO pins can supply only 3.3V, so a 12V external power source is used to trigger the lock with the help of a relay.

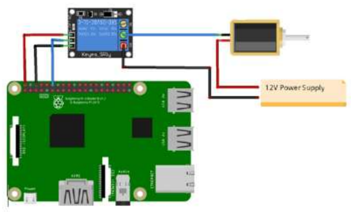


Fig12

4.14. Send Email

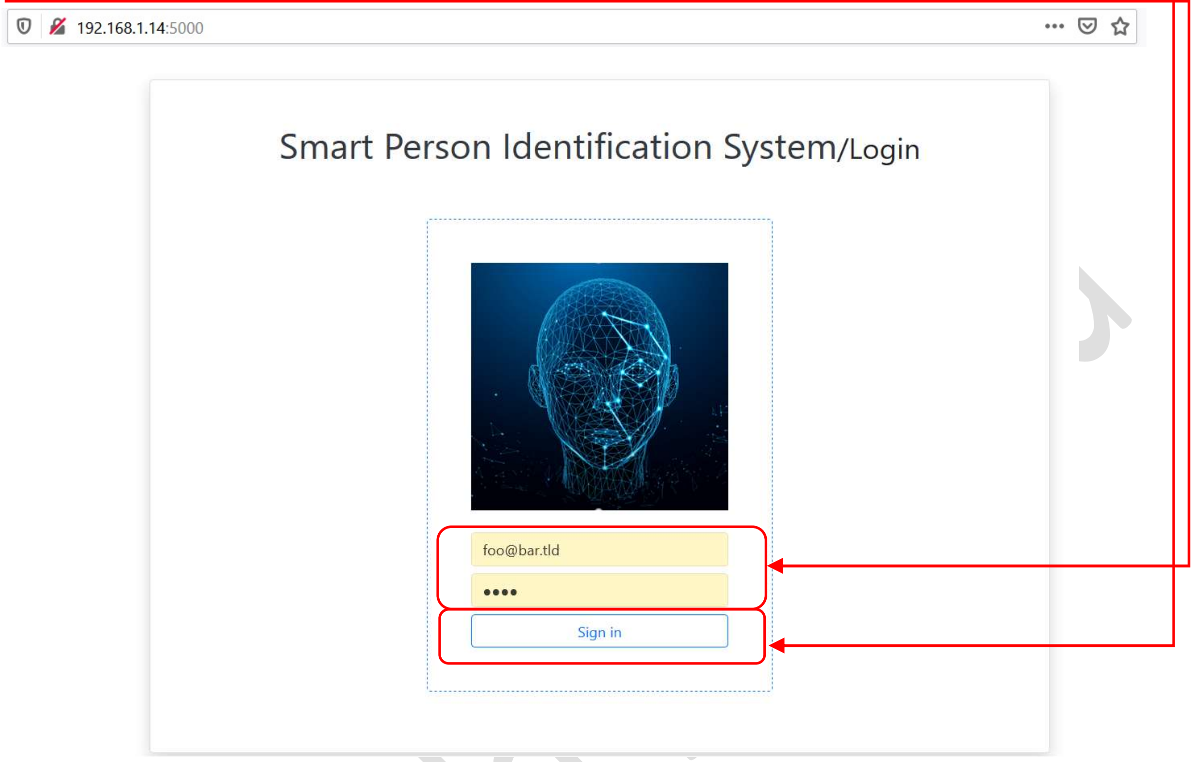
The procedure behind the email flow on the internet is SMTP (Simple Mail Transfer Protocol). The process of email delivery is quite similar to classical mail.
An organized system takes care of your electronic envelope and through a series of steps it drops it off to your recipient. In this process, the postman is an SMTP server which is simply a computer running SMTP, which takes care of delivering emails to their recipients.
There are various tools for sending an E-mail from Raspberry Pi using Command Line Interface, with their advantages and disadvantages.
In the project I implement E-mail sending by SMTP lib and Gmail account.
The SMTP lib, mime types, email and email, mime, application
library modules have been imported.
The function defined is named send Email(x). It has an integer argument x which is used to specify the recipient of the email.
The value 1 indicates the primary host and
The value 2 denotes the secondary host.
The directory containing the file saved is set as "/home/pi/ ".
This function uses the file "person.jpg" as an attachment.
The subject of the email is assigned the string "Someone at the door".
The body of the email message contains the text "The image of the person at the door is attached".

5. **Operating instructions**

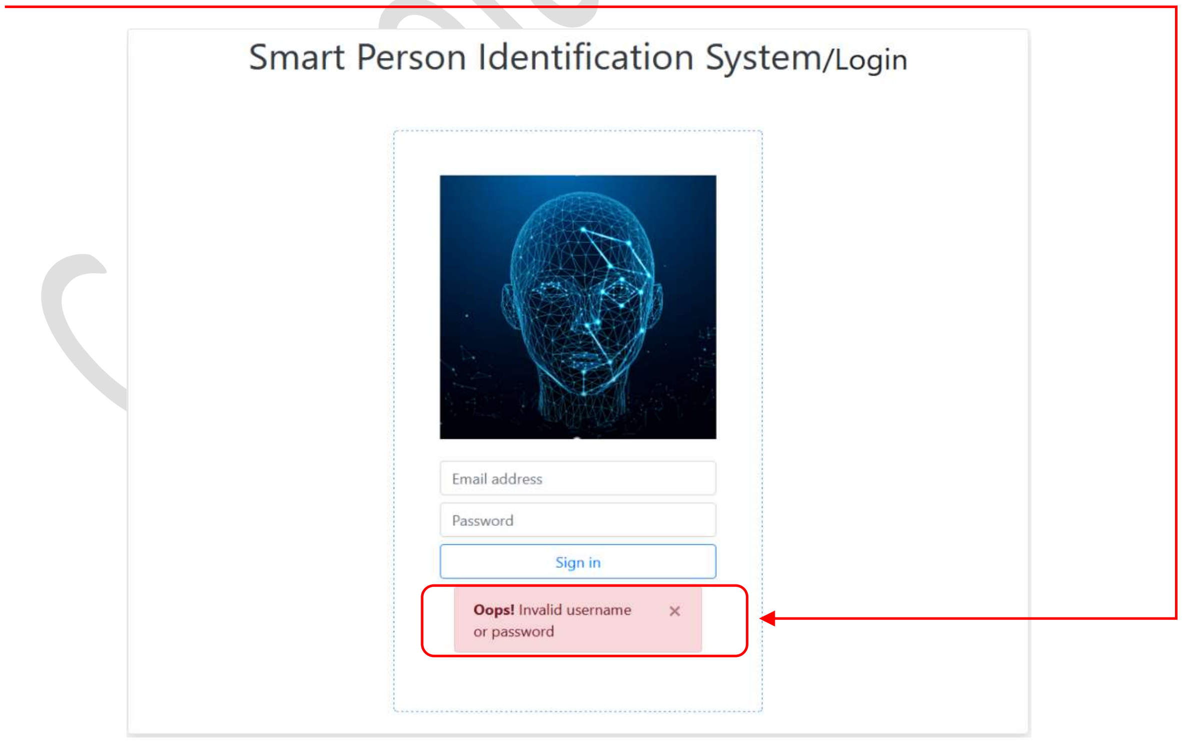
5.1. Web Application

5.1.1. Login page

Only a registered User (with email and password on Data base) can log in to the Web-Application. Before registering a new user, the system is received with an initial email and password.

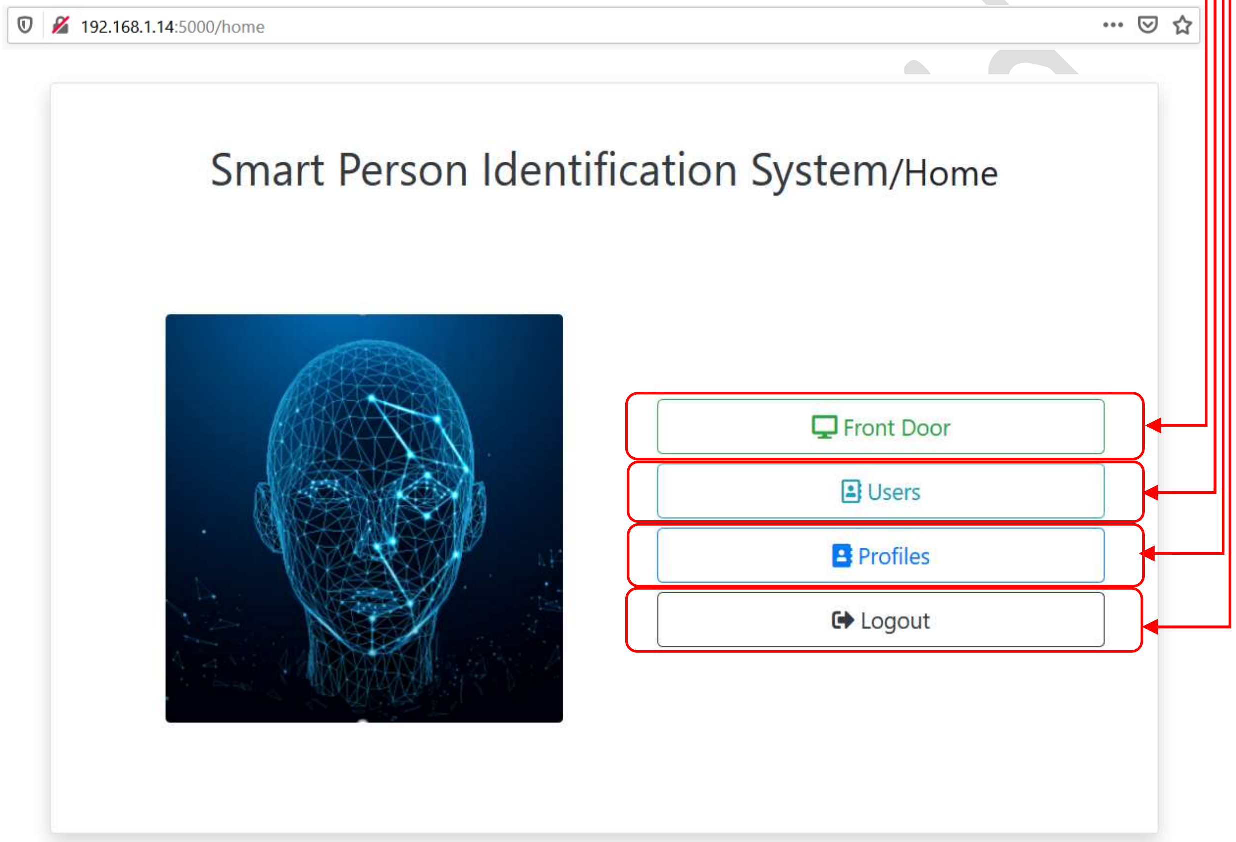


If you try to enter an incorrect username or password, an error alert will pop up.


Smart Person Identification System/Login
Email address
Password
Sign in
Oops! Invalid username or password ×

5.1.2. Home page

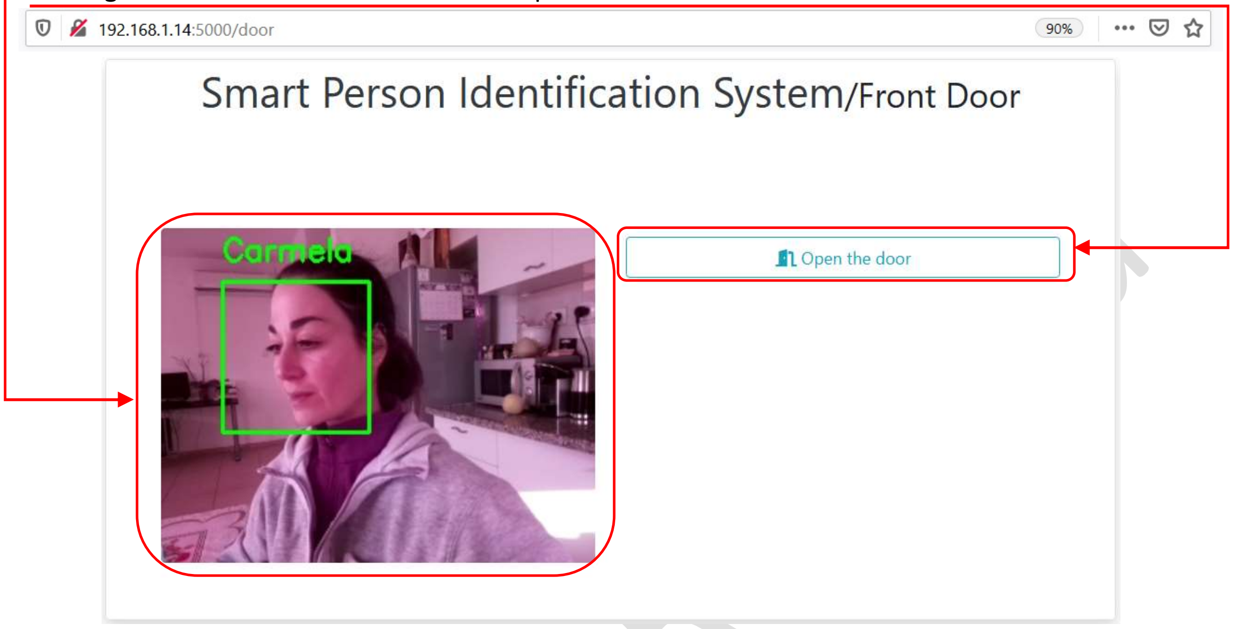
The first page after doing Login.
1. The first option it view camera of the front door And the possibility to let guests into the building.
2. Register a new user on the system (a registered user has an email and password, with which he can log in to the application).
3. Register a new profile on the system (The profile can be identified at the entrance of the door. For this, a profile is needed: name and sets of face images).
4. The fourth option; Secure exit from the system.



5.1.3. Front Door page

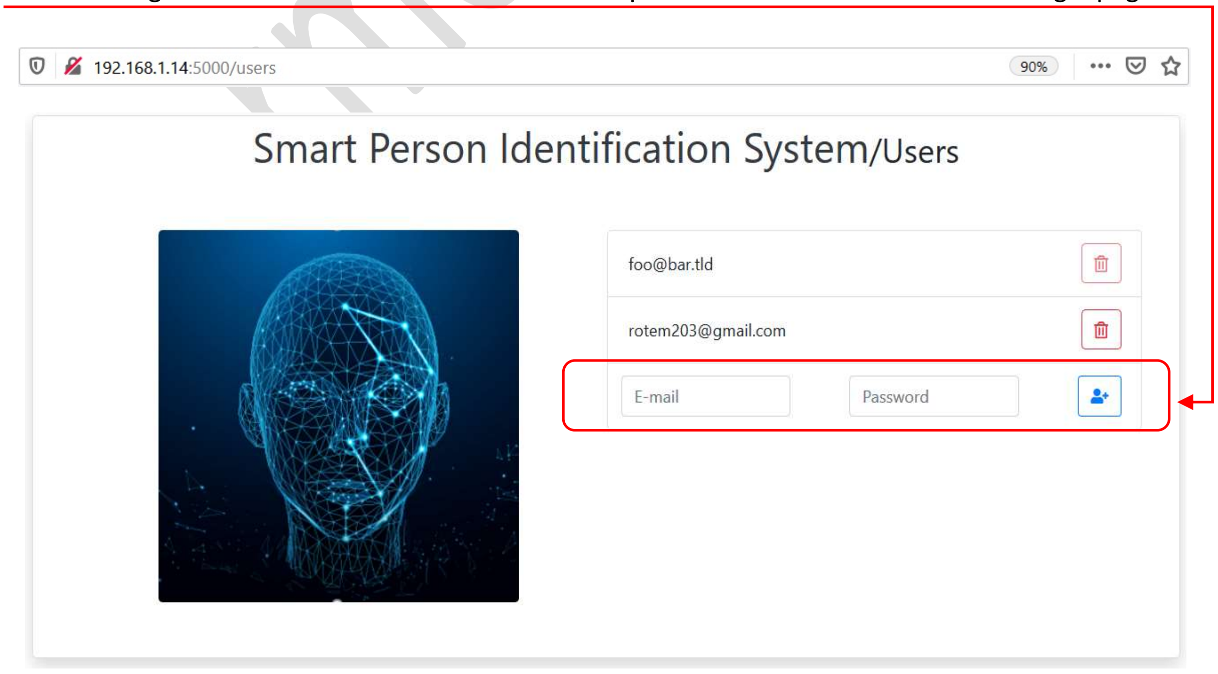
In this page you can check who is behind the door, allow him to enter (open the door) or send him a message.

Camera output
Pressing the button will allow the door to open



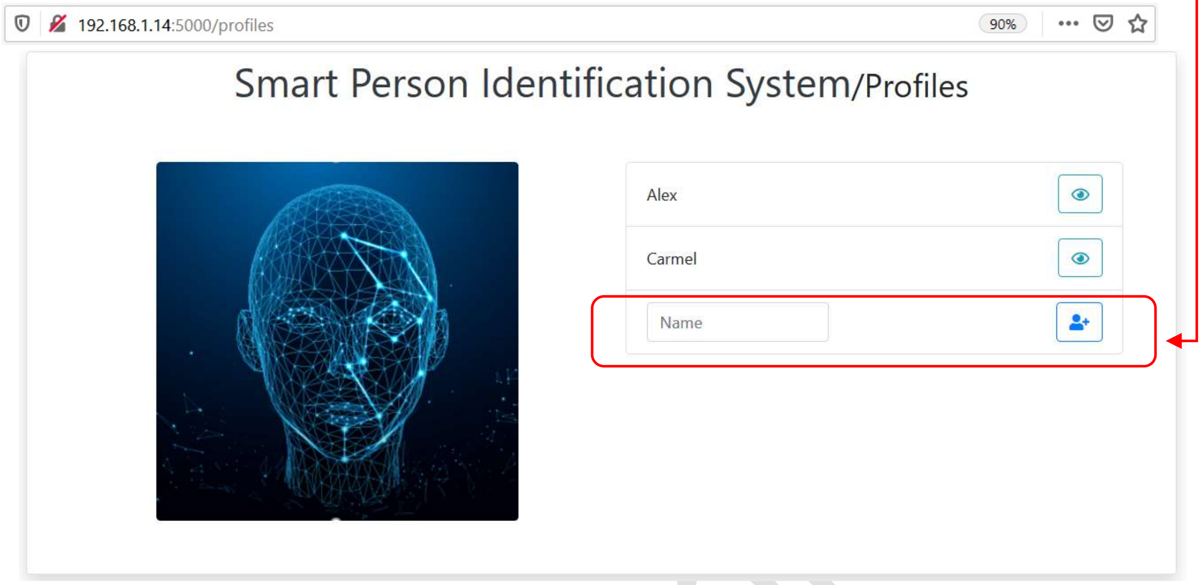
5.1.4. User list page

View existing users in the information system or register a new user.
New user registration: You need to set an email and password to identification on the login page.



5.1.5. Profiles list page

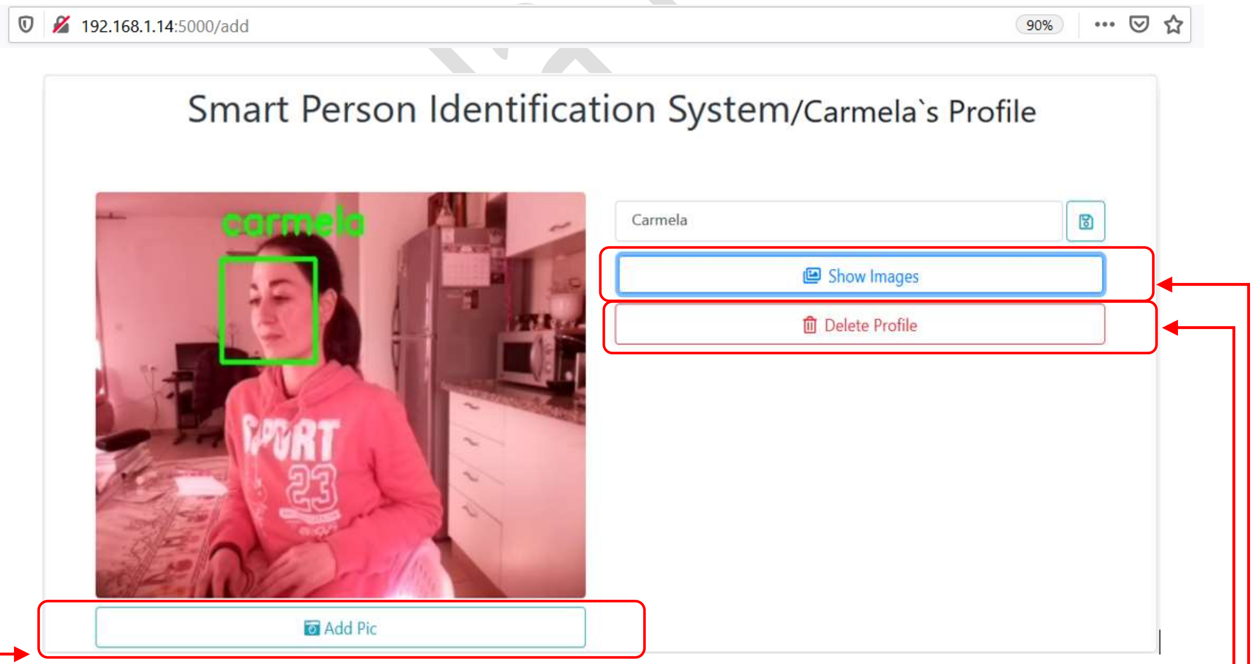
View existing profiles into Data base of the system or another option to register a new profile. Register a new profile: you need to set a name and go to the next page where inserting face Images of the profile. after that, Profile can be identified by the system and enters to the building.



192.168.1.14:5000/profiles

Smart Person Identification System/Profiles

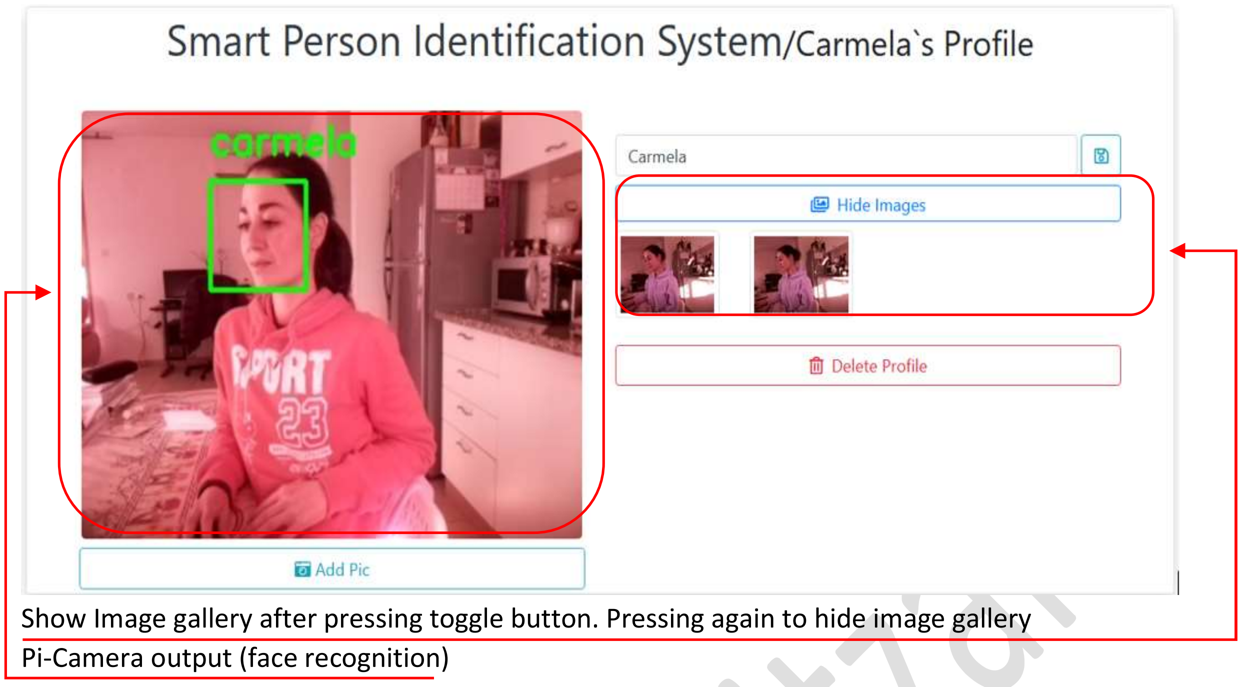
5.1.6. Add new Profile page (After pressing to Add Profile, in Profile page)



192.168.1.14:5000/add

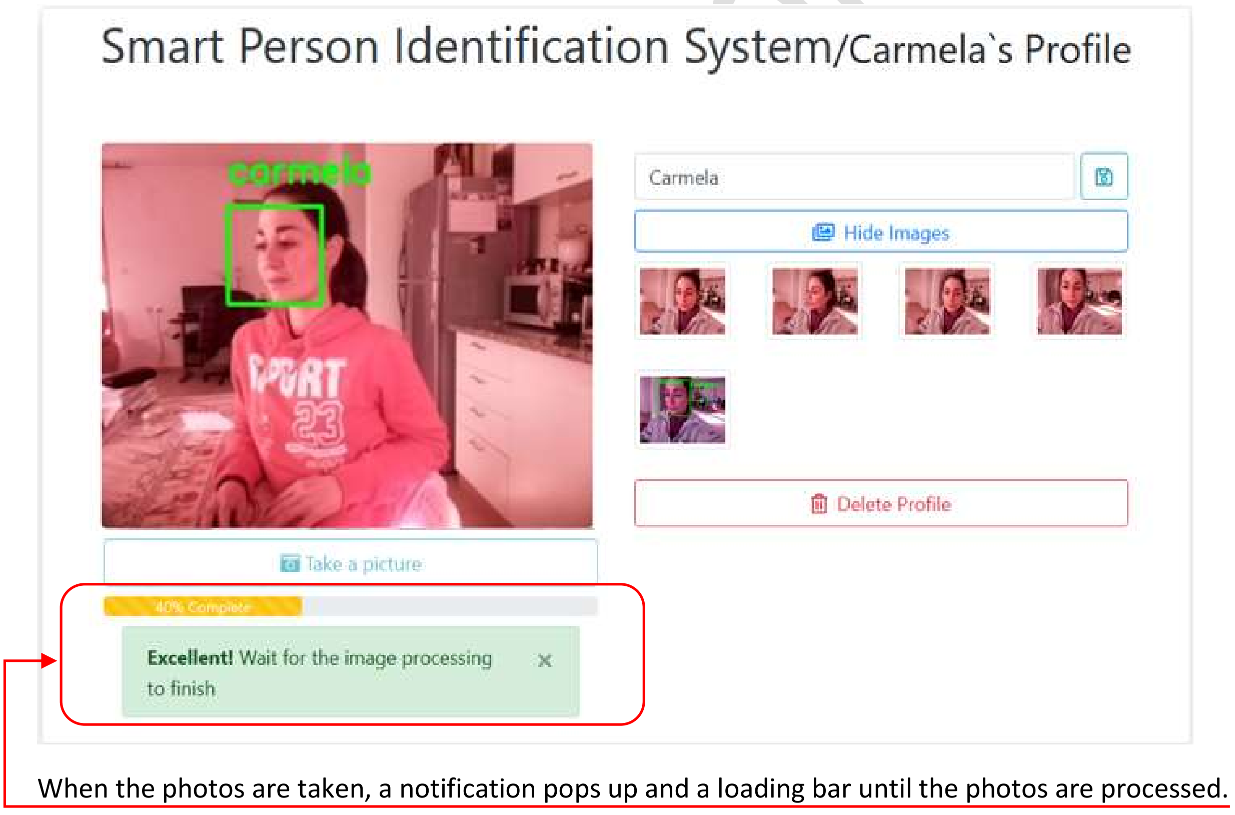
Smart Person Identification System/Carmela`s Profile

Show Image gallery by pressing toggle button
Take a photo by pressing a button (Add Pic).
The captured image enters the system database (up to 6 images must be taken)
The photographed image is displayed in the photo gallery.
Delete Profile from Application (Delete from Data base)



Show Image gallery after pressing toggle button. Pressing again to hide image gallery

Pi-Camera output (face recognition)



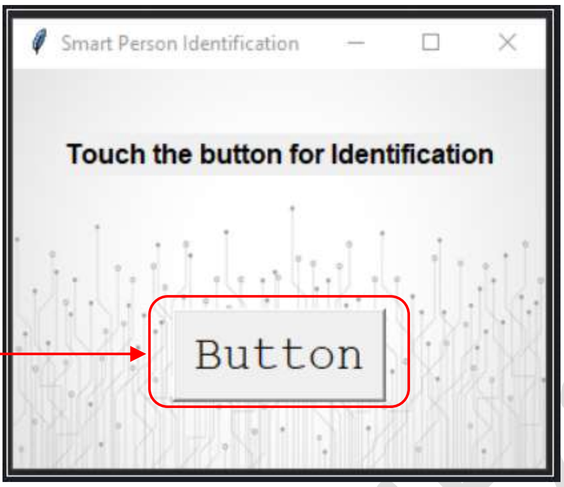
Excellent! Wait for the image processing to finish

When the photos are taken, a notification pops up and a loading bar until the photos are processed.

5.2. Touch Screen GUI

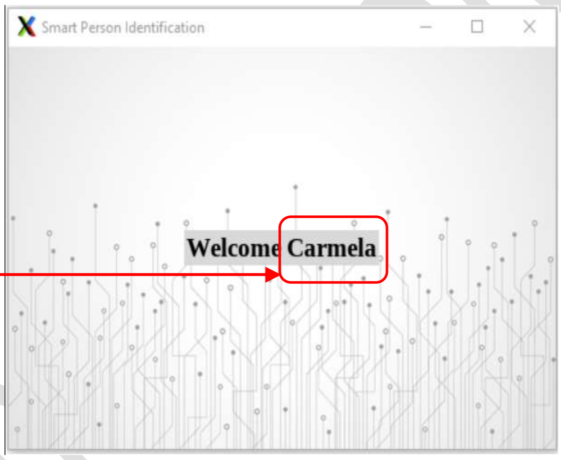
5.2.1. Visitor/ Worker touch screen

Exterior screen (outside the front door).



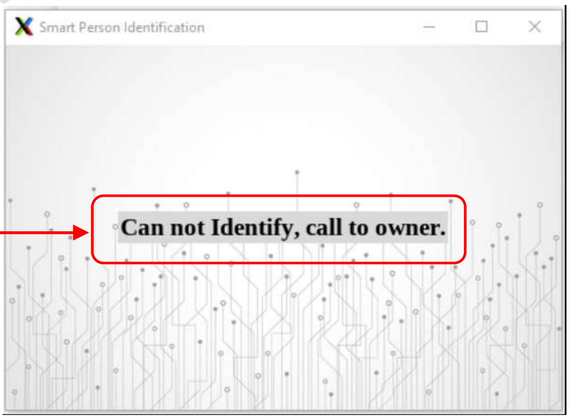
Touch the button for Identification

The visitor presses the button, and the system performs identification.



Welcome Carmela

When successfully identified, a greeting message and the **name** of the identified profile will appear.



Can not Identify, call to owner.

When the identification process fails, an appropriate message will appear.

6. Software Engineering Process

6.1. Requirements and Use-Cases

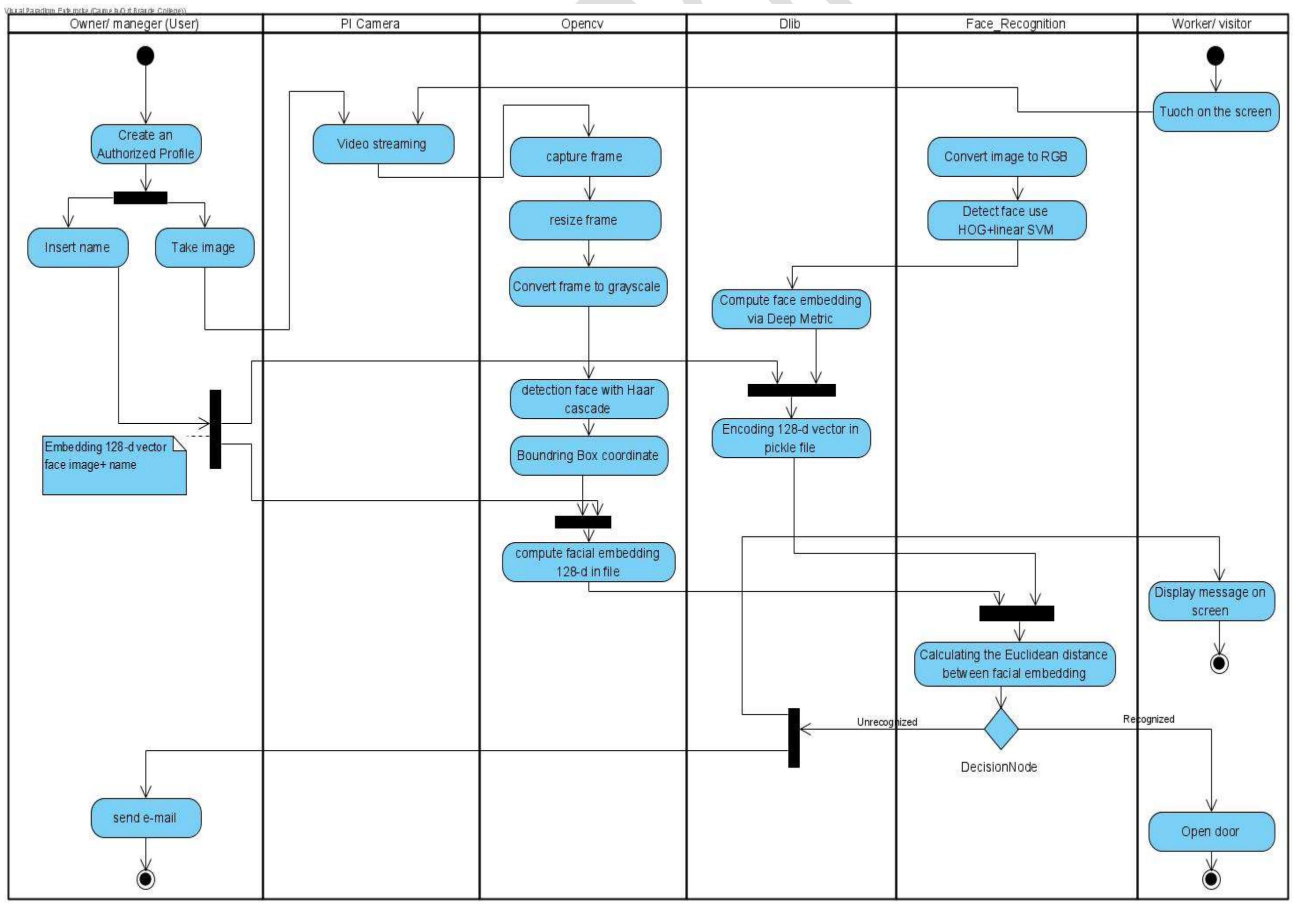
ID	Requirement
1	The System will recognize human.
2	The Recognized person photograph will be converted into list of 128 real-valued numbers.
3	The system will use the Pi- camera to analyze in real time.
4	The detection will be done with Haar cascade.
5	The detection will be done with HOG and SVM linear.
6	The system will take a photo of the person.
7	The System will be sending Email to owner.
8	The recognition will be done with ResNet-34 network.
9	The Trainer can re-train the model.

Use Case	**Face Analyzing**
Actors and Goals	Office worker/ Visitor: Receive details of the identification process: successful or failed.
Pre-conditions	The camera is working
Post-conditions	The faces were correctly recognizing
Trigger	The Office worker/ Visitor clicks on the touch button.
Main Success Scenario	The Office worker/ Visitor clicks on the touch button. The system analyzes the video. The system outputs the recognized face.
Branching	-
Further Requirements	The system will be taking a photo of face person. The system will be sending photo of unrecognized person with data and timing.

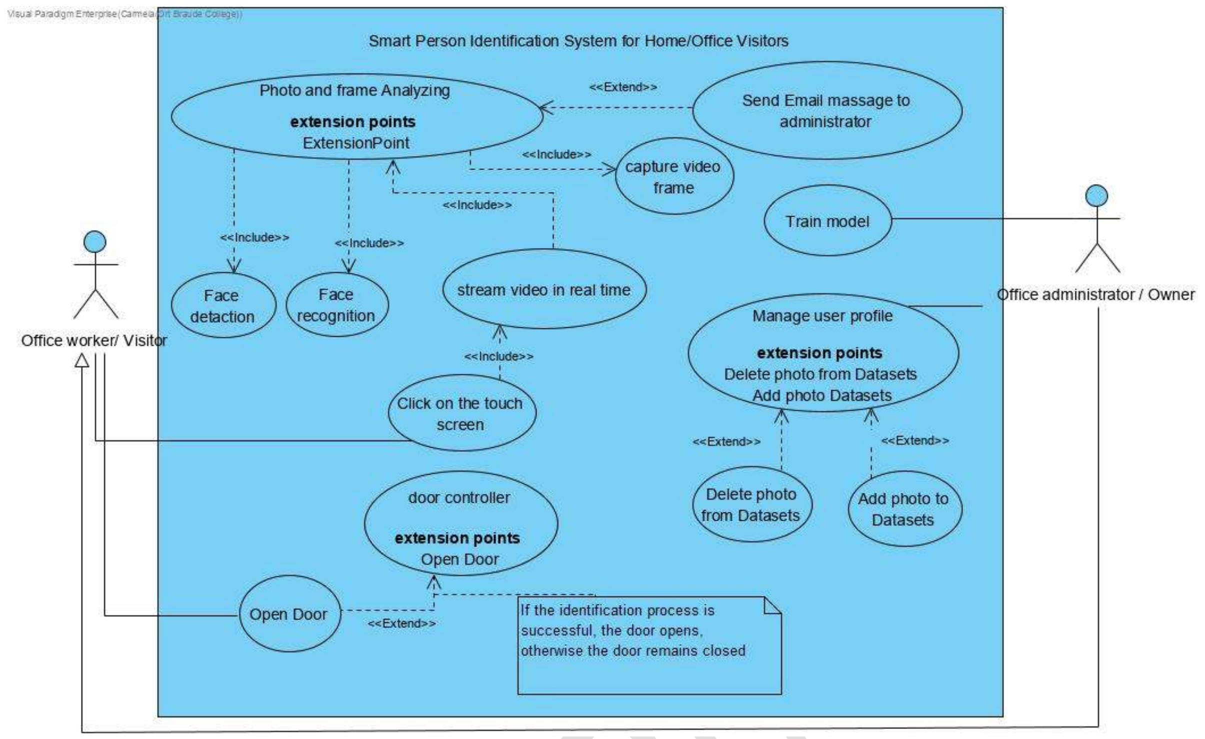
Use Case	Model Training
Actors and Goals	Office administrator / Owner: Train the Model.
Pre-conditions	Dataset is ready.
Post-conditions	Model is trained.
Trigger	The Trainer runs the training command.
Main success scenario	The system is training the Model. The Model is trained.
Branching	-
Further Requirements	-

7. Maintenance Guide and UML Diagrams

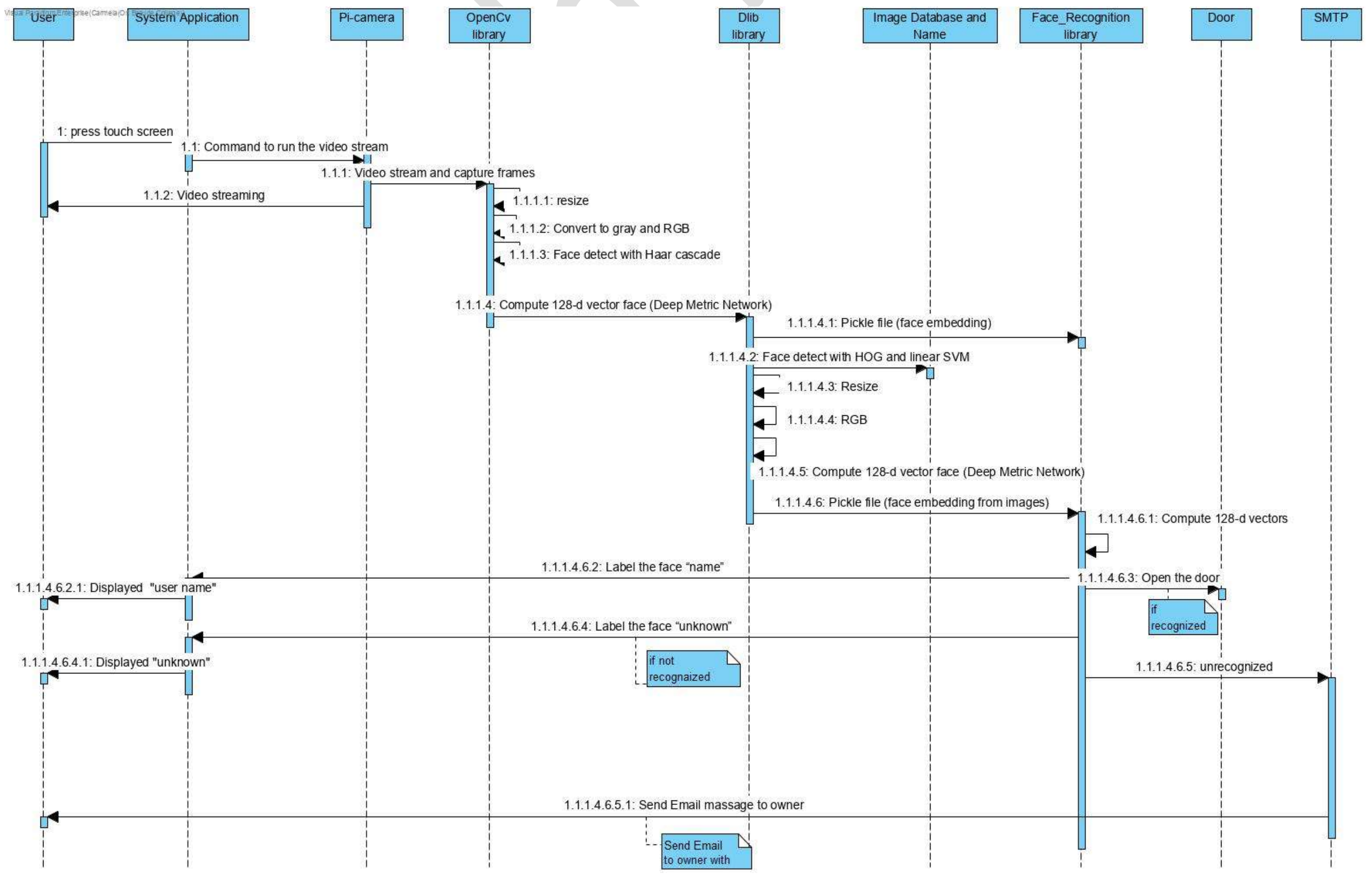
7.1. Activity Diagram


Owner/ maneger (User)
PI Camera
Opencv
Dlib
Face_Recognition
Worker/ visitor
Create an Authorized Profile
Insert name
Take image
Embedding 128-d vector face image+ name
Video streaming
capture frame
resize frame
Convert frame to grayscale
detection face with Haar cascade
Boundring Box coordinate
compute facial embedding 128-d in file
Compute face embedding via Deep Metric
Encoding 128-d vector in pickle file
Convert image to RGB
Detect face use HOG+linear SVM
Calculating the Euclidean distance between facial embedding
DecisionNode
Unrecognized
Recognized
Tuoch on the screen
Display message on screen
Open door
send e-mail

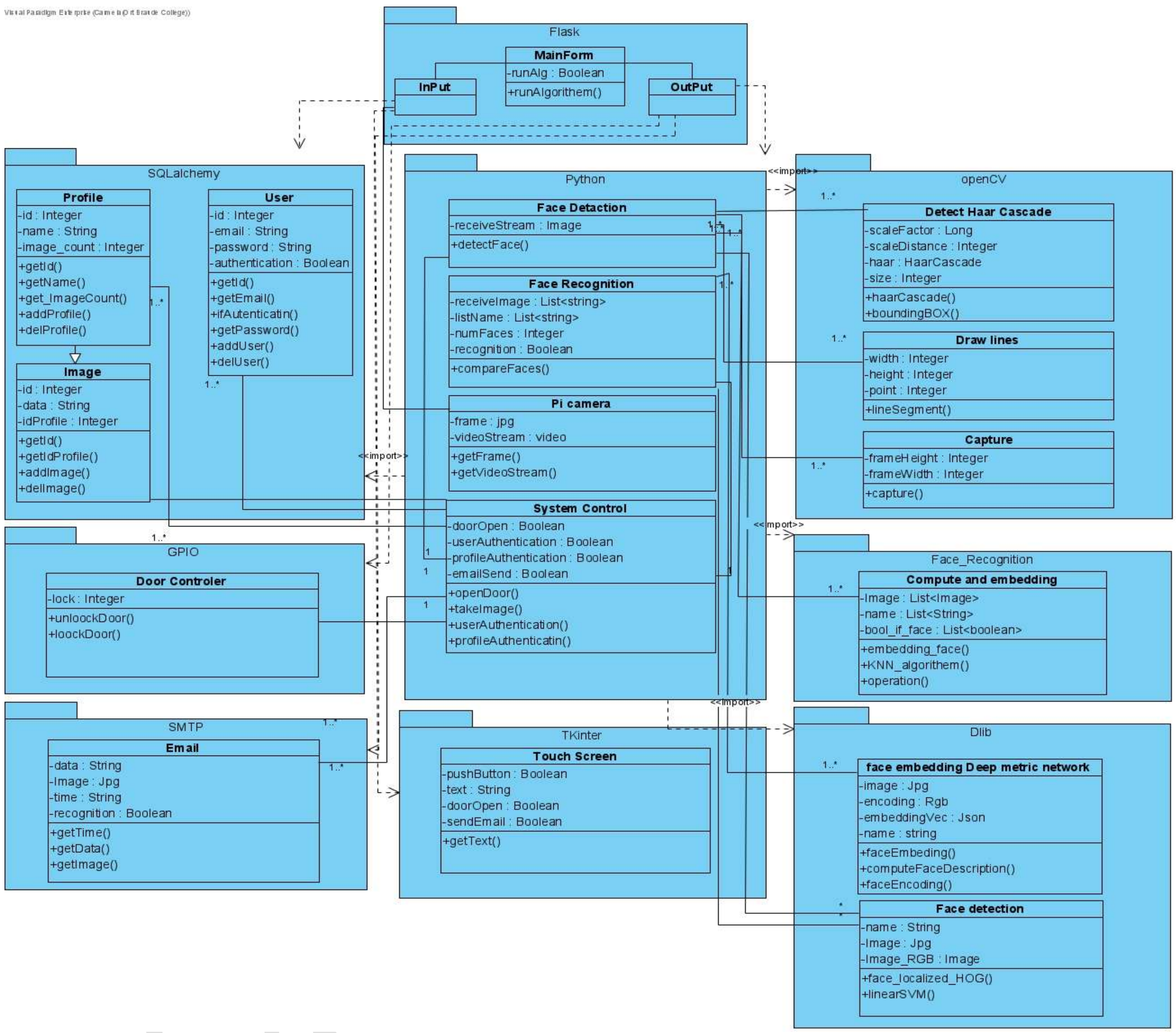
7.2. Use Case Diagram

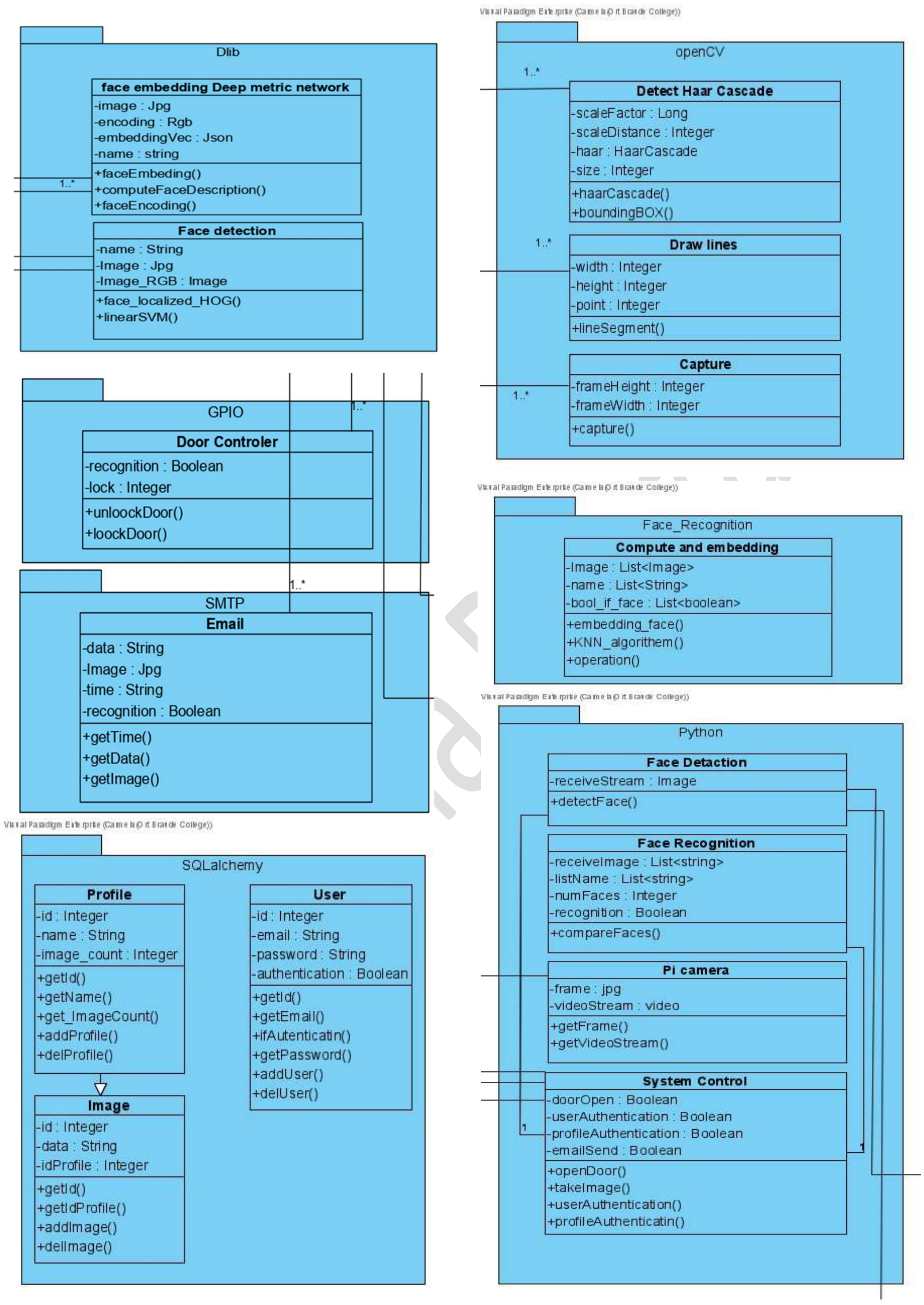

Visual Paradigm Enterprise(Carmela(Ort Braude College))
Smart Person Identification System for Home/Office Visitors
Photo and frame Analyzing
extension points
ExtensionPoint
<<Extend>>
Send Email massage to administrator
<<Include>>
capture video frame
<<Include>>
Train model
<<Include>>
<<Include>>
Office administrator / Owner
Face detaction
Face recognition
stream video in real time
Office worker/ Visitor
Manage user profile
extension points
Delete photo from Datasets
Add photo Datasets
<<Include>>
Click on the touch screen
<<Extend>>
<<Extend>>
Delete photo from Datasets
Add photo to Datasets
door controller
extension points
Open Door
Open Door
<<Extend>>
If the identification process is successful, the door opens, otherwise the door remains closed

7.3. Sequence Diagram

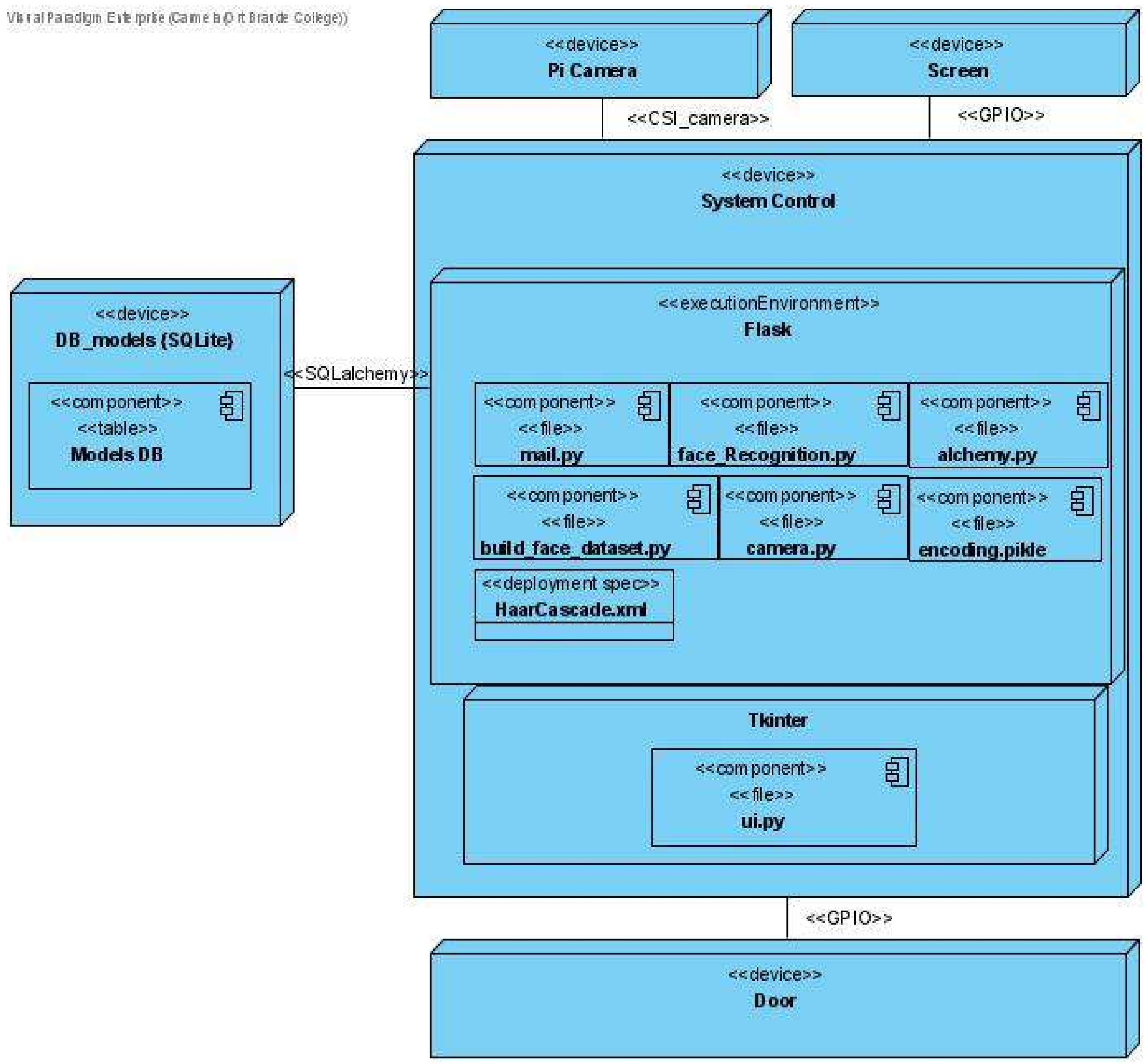

User
System Application
Pi-camera
OpenCv library
Dlib library
Image Database and Name
Face_Recognition library
Door
SMTP
1: press touch screen
1.1: Command to run the video stream
1.1.1: Video stream and capture frames
1.1.2: Video streaming
1.1.1.1: resize
1.1.1.2: Convert to gray and RGB
1.1.1.3: Face detect with Haar cascade
1.1.1.4: Compute 128-d vector face (Deep Metric Network)
1.1.1.4.1: Pickle file (face embedding)
1.1.1.4.2: Face detect with HOG and linear SVM
1.1.1.4.3: Resize
1.1.1.4.4: RGB
1.1.1.4.5: Compute 128-d vector face (Deep Metric Network)
1.1.1.4.6: Pickle file (face embedding from images)
1.1.1.4.6.1: Compute 128-d vectors
1.1.1.4.6.2: Label the face "name"
1.1.1.4.6.3: Open the door
1.1.1.4.6.2.1: Displayed "user name"
if recognized
1.1.1.4.6.4: Label the face "unknown"
1.1.1.4.6.4.1: Displayed "unknown"
if not recognaized
1.1.1.4.6.5: unrecognized
1.1.1.4.6.5.1: Send Email massage to owner
Send Email to owner with

7.4. Class and Package Diagram

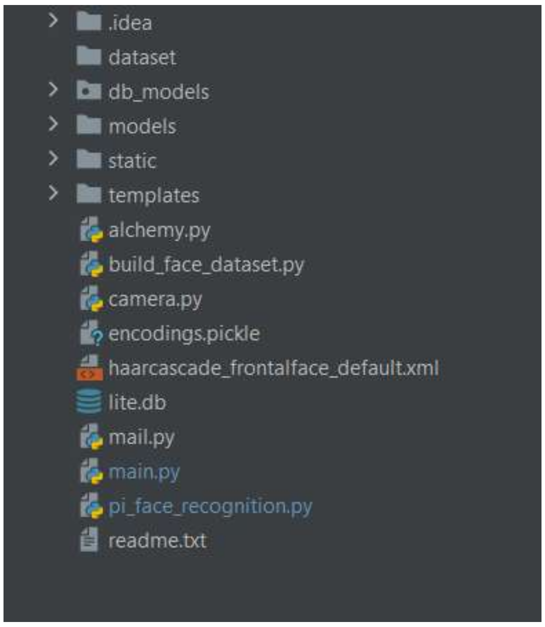



Dlib
face embedding Deep metric network
-image : Jpg
-encoding : Rgb
-embeddingVec : Json
-name : string
+faceEmbeding()
+computeFaceDescription()
+faceEncoding()
Face detection
-name : String
-Image : Jpg
-Image_RGB : Image
+face_localized_HOG()
+linearSVM()
1..*
GPIO
Door Controler
-recognition : Boolean
-lock : Integer
+unloockDoor()
+loockDoor()
1..*
SMTP
Email
-data : String
-Image : Jpg
-time : String
-recognition : Boolean
+getTime()
+getData()
+getImage()
1..*
SQLalchemy
Profile
-id : Integer
-name : String
-image_count : Integer
+getId()
+getName()
+get_ImageCount()
+addProfile()
+delProfile()
Image
-id : Integer
-data : String
-idProfile : Integer
+getId()
+getIdProfile()
+addImage()
+delImage()
User
-id : Integer
-email : String
-password : String
-authentication : Boolean
+getId()
+getEmail()
+ifAutenticatin()
+getPassword()
+addUser()
+delUser()
openCV
1..*
Detect Haar Cascade
-scaleFactor : Long
-scaleDistance : Integer
-haar : HaarCascade
-size : Integer
+haarCascade()
+boundingBOX()
1..*
Draw lines
-width : Integer
-height : Integer
-point : Integer
+lineSegment()
1..*
Capture
-frameHeight : Integer
-frameWidth : Integer
+capture()
Face_Recognition
Compute and embedding
-Image : List<Image>
-name : List<String>
-bool_if_face : List<boolean>
+embedding_face()
+KNN_algorithem()
+operation()
Python
Face Detaction
-receiveStream : Image
+detectFace()
Face Recognition
-receiveImage : List<string>
-listName : List<string>
-numFaces : Integer
-recognition : Boolean
+compareFaces()
Pi camera
-frame : jpg
-videoStream : video
+getFrame()
+getVideoStream()
System Control
-doorOpen : Boolean
-userAuthentication : Boolean
-profileAuthentication : Boolean
-emailSend : Boolean
+openDoor()
+takeImage()
+userAuthentication()
+profileAuthenticatin()
1
1

6.5. Deployment Diagram



8. Project structure



```
> .idea
  dataset
> db_models
> models
> static
> templates
  alchemy.py
  build_face_dataset.py
  camera.py
  encodings.pickle
  haarcascade_frontalface_default.xml
  lite.db
  mail.py
  main.py
  pi_face_recognition.py
  readme.txt
```

dataset/: This directory should contain sub-directories for each person you would like your facial recognition system to recognize.

db_model/: Create Data base (tables) for main structure: Images (profiles faces), profiles (how can identify by the System, Users (how can manage the web-application.

templates/: All HTML pages for web-application.

encodings.pickle: Our face encodings (128-d vectors, one for each face) are stored in this pickle file.

static/: Responsible for the visibility of the web-application.

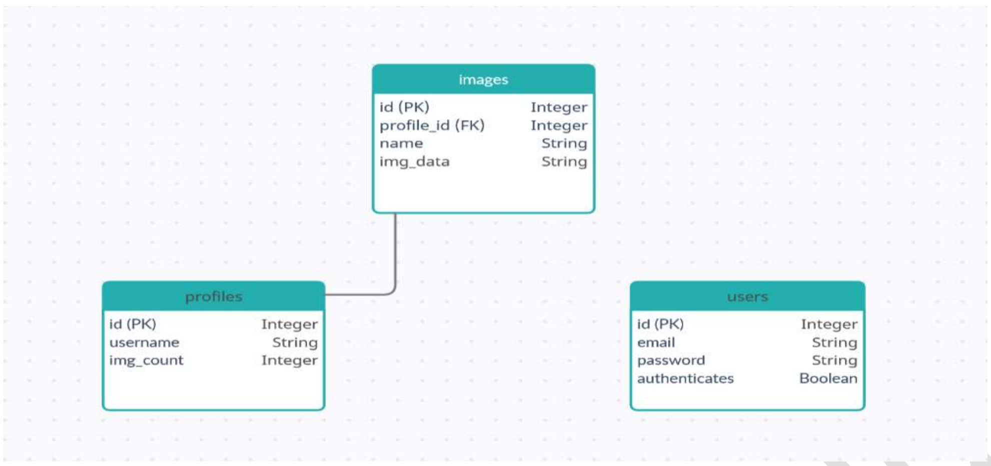
main.py: The main file, menage for the web-application (Write in Flask).

haarcascade_frontalface_default.xml: In order to detect and localize faces in frames it OpenCV's pre-trained Haar cascade file.

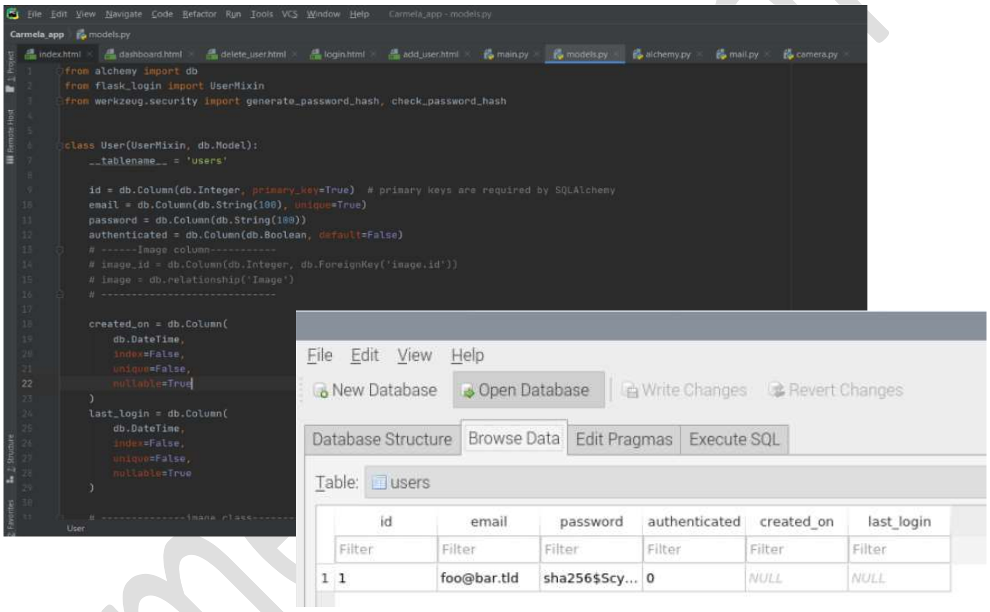
pi_face_recognition.py: This is our main execution script. In this script we will use OpenCV's Haar cascade to *detect and localize* the face. From there, we will continue with the same method to actually *recognize* the face next to find faces in our dataset and encode them into 128-d vectors.

mail.py: Responsible for commands for sending email by the system with a photo attached

8.1. Build Data base with SQLalchemy and SQLite



8.1.1. Users table

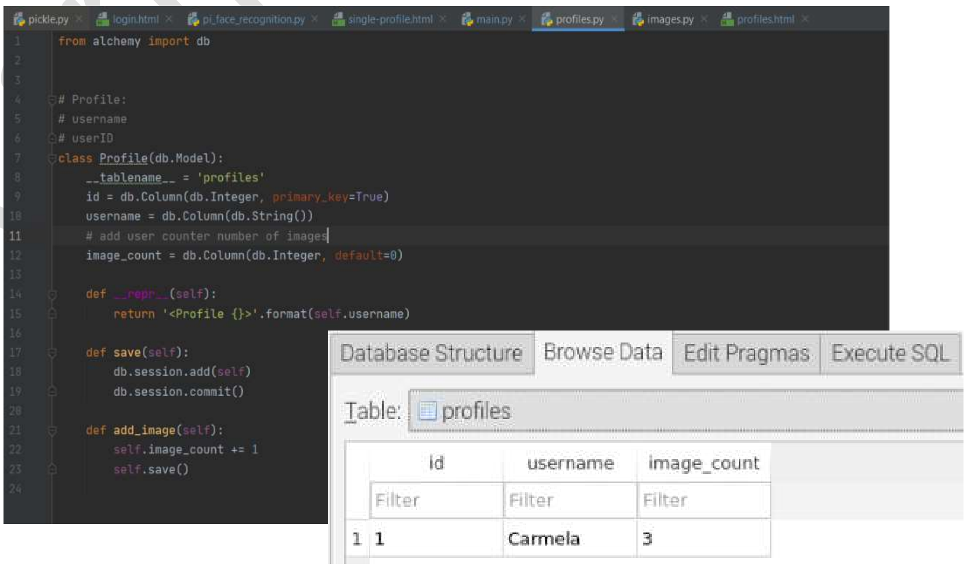


```python
from alchemy import db
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash


class User(UserMixin, db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)  # primary keys are required by SQLAlchemy
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    authenticated = db.Column(db.Boolean, default=False)
    # ------Image column-----------
    # image_id = db.Column(db.Integer, db.ForeignKey('image.id'))
    # image = db.relationship('Image')
    # -----------------------------

    created_on = db.Column(
        db.DateTime,
        index=False,
        unique=False,
        nullable=True
    )
    last_login = db.Column(
        db.DateTime,
        index=False,
        unique=False,
        nullable=True
    )

    # ---------------image class---------
```

Table: users

	id	email	password	authenticated	created_on	last_login
1	1	foo@bar.tld	sha256$Scy…	0	NULL	NULL

8.1.2. Profiles table

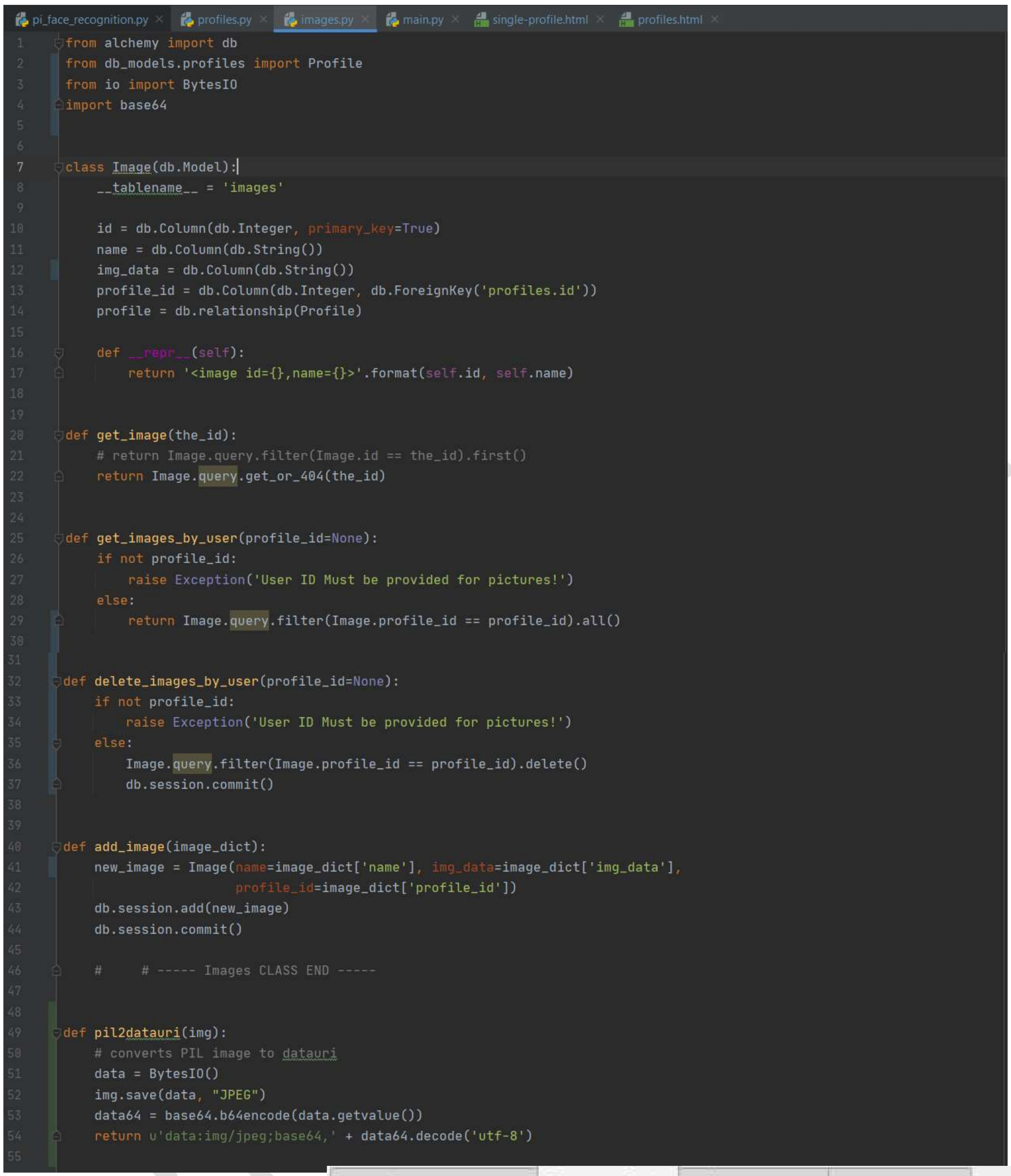


```python
from alchemy import db


# Profile:
# username
# userID
class Profile(db.Model):
    __tablename__ = 'profiles'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String())
    # add user counter number of images
    image_count = db.Column(db.Integer, default=0)

    def __repr__(self):
        return '<Profile {}>'.format(self.username)

    def save(self):
        db.session.add(self)
        db.session.commit()

    def add_image(self):
        self.image_count += 1
        self.save()

```

Table: profiles

	id	username	image_count
1	1	Carmela	3

8.1.3. Images table



```python
from alchemy import db
from db_models.profiles import Profile
from io import BytesIO
import base64


class Image(db.Model):
    __tablename__ = 'images'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String())
    img_data = db.Column(db.String())
    profile_id = db.Column(db.Integer, db.ForeignKey('profiles.id'))
    profile = db.relationship(Profile)

    def __repr__(self):
        return '<image id={},name={}>'.format(self.id, self.name)


def get_image(the_id):
    # return Image.query.filter(Image.id == the_id).first()
    return Image.query.get_or_404(the_id)


def get_images_by_user(profile_id=None):
    if not profile_id:
        raise Exception('User ID Must be provided for pictures!')
    else:
        return Image.query.filter(Image.profile_id == profile_id).all()


def delete_images_by_user(profile_id=None):
    if not profile_id:
        raise Exception('User ID Must be provided for pictures!')
    else:
        Image.query.filter(Image.profile_id == profile_id).delete()
        db.session.commit()


def add_image(image_dict):
    new_image = Image(name=image_dict['name'], img_data=image_dict['img_data'],
                      profile_id=image_dict['profile_id'])
    db.session.add(new_image)
    db.session.commit()

    #     # ----- Images CLASS END -----


def pil2datauri(img):
    # converts PIL image to datauri
    data = BytesIO()
    img.save(data, "JPEG")
    data64 = base64.b64encode(data.getvalue())
    return u'data:img/jpeg;base64,' + data64.decode('utf-8')
```

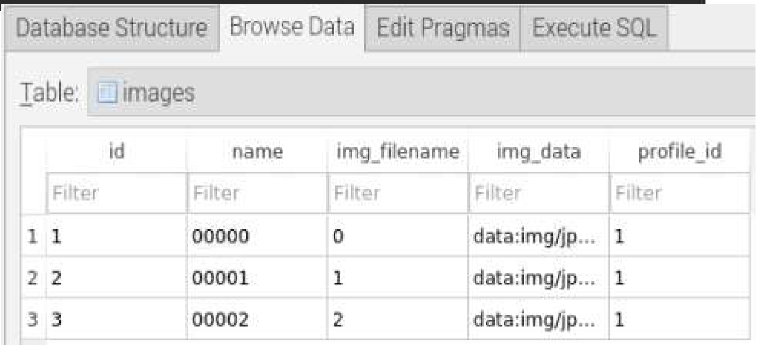


Table: images

	id	name	img_filename	img_data	profile_id
	Filter	Filter	Filter	Filter	Filter
1	1	00000	0	data:img/jp...	1
2	2	00001	1	data:img/jp...	1
3	3	00002	2	data:img/jp...	1

9. The work environment of the project

In terms of **hardware**:

1. Raspberry Pi
2. Pi-Camera
3. LCD display
4. Servo motor

In terms of **software**:

1. The project is based on face recognition and face detection and for that I use libraries like: OpenCV, cv2, face_recognition, Dlib.

2. Also, the project is written in Python so make sure Python 3.7 exists. The frame work is FLASK so it should be installed on the rasppbery pi.
3.
4. Also, for app design I used Bootstrap.

To build the external GUI, I used TKinter.

9.1. Installation instructions

Install prerequisites on your Raspberry Pi
The Raspberry Pi requires that you install a few system packages before you get started:

```
pip install opencv
```

```
$ sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-100
$ sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
$ sudo apt-get install libatlas-base-dev
$ sudo apt-get install libjasper-dev
```

Install pip on your Raspberry Pi
The Python package manager, “pip”, can be obtained via wget:

```
pip install opencv
```

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python3 get-pip.py
```

Install OpenCV into a virtual environment on your Raspberry Pi with pip

```
pip install opencv
```

```
$ pip install virtualenv virtualenvwrapper
```

Then you need to add the following lines to your `~/.bashrc` . Open the file using `nano` , `vim` , or `emacs` and append these lines to the end:

```
pip install opencv
```

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

Save the file. Then “source it” in your terminal:

pip install opencv

```
$ source ~/.bashrc
```

To create a Python 3 virtual environment which will house OpenCV and other packages you install, simply use mkvirtualenv and the command below:

pip install opencv

```
$ mkvirtualenv cv -p python3
```

Now you have a virtual environment named `cv` . You can activate it any time via:

pip install opencv

```
$ workon cv
```

And now with a flip of the wrist, you can **pip install OpenCV** into `cv` :

pip install opencv

```
$ pip install opencv-contrib-python==4.1.0.25
```

using the PiCameraYou can install the Python module using the following command:

pip install opencv

```
$ pip install "picamera[array]"
```

Next, let us install Davis King's dlib toolkit software into the same Python virtual environment (provided you are using one) that you installed OpenCV into:

Raspberry Pi Face Recognition

```
$ workon <your env name> # optional
$ pip install dlib
```

From there, simply use pip to install Adam Geitgey's face_recognition module:

Raspberry Pi Face Recognition

```
$ workon <your env name> # optional
$ pip install face_recognition
```

Install imutils package of convenience functions:

Raspberry Pi Face Recognition

```
$ workon <your env name> # optional
$ pip install imutils
```

Install Flask Micro Framework

OpenCV – Stream video to web browser/HTML page

```
$ pip install flask
```

Install SQLalchemy

```
$ pip install flask-sqlalchemy
```

10. Test Process

To check out the system performance and accuracy I ran the program on some significant inputs and tests.

Test No.	Test subject	Expected result	Result
1.	User added "new profile"	The system expects to receive a name and face photos of the profile	Pass
2.	Frame of an unknown person	Message of "Unknown". Send Email to owner.	Pass
3.	frame of a known person	Show “Name person” on a display and open the door.	Pass
4.	Input an image without face person	The system will not recognize the new profile added.	Pass
5.	The user tray to login with incorrect password or email	The web application will pops up error message	Pass
8.	Analyze different unknown people.	Message of "Unknown". Send Email.	Pass
9.	The owner of the system adds a new person with permission	The person is successfully added	Pass
10.	The owner press “User permit"	The new user can login the system	pass

11. Project Review and Process Description

11.1. Description of the development process:

To build the system based on Face Detection and Face Recognition, I had to examine the issues in depth by reading related articles and looking at all the different ways of realization. I learned about different types of Face recognition exercises and while I had to consider the project is installed in a Linux operating system on Raspberry Pi. Raspberry Pi is a mini-computer, and its graphics capabilities are not high, as is the size of the memory. I chose to implement HOG using a waterfall classification and deep metric learning for face recognition that uses a few memory resources and a graphics card to install on Raspberry Pi. Besides, I focused on building a GUI system for profiles and views for the system user. To perform this, I had to learn to use Python, HTML/CSS, and Flask.

11.2. Description of the testing process:

For the test of the system, I used manual testing. That is, every step in developing the system from capturing faces and Face Detection and Recognition process, I tested the identification process on different faces of people and on animals. I found that faces inserted into the Database of the system were identified in significant percentages compared to faces that are not inserted into the Database of the system. I found that there is a limitation to the system when dividing the resource (video feed) between different screens, so I tried producing a system that divides this resource safely. Also, I checked the login process of the users by providing different passwords or emails, and I also developed a secure login and logout process, so that the user cannot navigate to one of the site pages without logging in to the system.

11.3. Results and conclusions:

At the point of choosing the face recognition algorithm, I encountered a problem that exists in Raspberry Pi, the limited memory, and limitations of its graphics card, so I had to adapt the face recognition algorithm to Raspberry Pi's capabilities.

In the experimental phase, I performed many performances on different image input and different limits of the algorithm, to better comprehend how the algorithm responds to the values of different variables. I learned the algorithm responded better to several groups. For example, the algorithm gives a less good result for images taken directly from the PI camera with which I also work for face recognition than for higher-quality facial images that were manually inserted into the system's information system. This difference is likely because the pixel density and image quality are higher in contrast to the photo on the Pi camera that some of the images are smeared and the image quality is poorer.

12. REFERENCES

[1] Mastering OpenCV 4 with Python- A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7 by Alberto Fernández Villán. Copyright © 2019 Packt Publishing

[2] Face Detection and Recognition in Video-Streams by Jannik Boll Nielsen. Kongens Lyngby 2010 IMM-B.Sc-2010-14

[3] Evolution of Convolutional Neural Network Architecture in Image Classification Problems by Andrey Arsenov1, Igor Ruban1, Kyrylo Smelyakov1, Anastasiya Chupryna1

[4] Deep Learning with Keras by Antonio Gulli and Sujit Pal. Copyright © 2017 Packt Publishing

[5] Learn Keras for Deep Neural Networks A Fast-Track Approach to Modern Deep Learning with Python by Jojo John Moolayil (z-lib.org)

[6] Python Deep Learning Projects by Matthew Lamons, Rahul Kumar, Abhishek Nagaraja (z-lib.org)

[7] Python Image Processing Cookbook (z-lib.org)

[8] Practical Machine Learning and Image Processing For Facial Recognition, Object Detection, and Pattern Recognition Using Python by Himanshu Singh (z-lib.org)

[9] Navneet Dalal, Bill Triggs, "Histograms of Oriented Gradients for Human Detection", INRIA Rhone-Alpes, 655 avenue de l'Europe, Montbonnot 38334, France.

[10] SHU Chang, DING Xiaoqing, FANG Chi, "Histogram of the Oriented Gradient for Face Recognition", TSINGHUA SCIENCE AND TECHNOLOGY, ISSN 1007-0214 15/15 pp216-224, Volume 16, Number 2, April 2011.