Integration Of Face Recognition Model to a Biometric Security System

Eggy Wirahman¹, Junita¹

¹Department of Electrical Engineering, Universitas Pelita Harapan

*For correspondence: ew80023@student.uph.edu

Abstract With the global COVID-19 pandemic at large, physical contact is supposed to be minimized. This raises the need for a non-invasive security system that could differentiate between individual human beings. Biometrics is the automatic recognition of an individual using certain distinguishing traits. It proves to be a solution for security systems in such times. However, computers cannot see objects as humans do, they see objects as binaries which requires further discoveries to build a system that is accurate enough to be implemented in a security system. Viola and Jones did research on such methods. They had found that computers are able to see patterns, but it requires a classifier which is a model to help computers understand certain features that makes a human face. Viola and Jones used what they call as the cascade classifiers with Haar features used by the computer to analyze an object and helps it determine whether the object is face or not. This research aims to create a multi-class classification (face or non-face and face in the database or not) facial recognition system with the help of machine learning to create a model which will implemented in the access systems. The OpenCV libraries are used in this research as there lots of facial recognition and processing functions which will render the research for designing machine learning based access systems easier. The research shows that the model formed has an accuracy of 74.8%, a precision of 74.02%, and a sensitivity of 100%.

Keywords: machine learning, cascade classifier, OpenCV, Raspberry Pi.

DOI: https://doi.org/xxxxx

Introduction

The era of the Fourth Industrial Revolution has begun. Machines no longer require the aid of humans in order for them to work and communicate with other machines. These days, machines can just communicate with other machines to execute the tasks they are designed to do [1]. It is all possible with the help of internet of things. People born in the time of the previous industrial revolution can only imagine the reach of the technology today.

The Fourth Industrial Revolution bring about the need for better security systems. Since device-to-device communications are possible and are being developed further at the moment, new security systems or protocols are also being developed. Biometrics are such security measures developed. Biometrics centers on the automatic recognition of an individual with the help of the individuals’ distinguishing traits [2].

Facial recognition is by method a part of biometrics. However, without human involvement, machines have a difficulty in differentiating between one human and another. The solution is to integrate the facial recognition system with an artificial intelligence to help them ‘learn’ the distinguishing features of each individuals’ faces. The artificial intelligence will be trained using machine learning, specifically the cascade classifier method.

This research aims to create access system with facial recognition by integrating the system with OpenCV’s pre-trained classifiers with enhanced training with machine learning. Raspberry Pi will be used to run the facial recognition program. This research will be limited to creating and training the machine learning model to recognize the features of faces and for the model to be able to distinguish between individual faces stored in the system.

The paper will discuss the theories used in the research which are cascade classifier and eigenfaces. The hardware Raspberry Pi, the library OpenCV, the confusion matrix table, and the accuracy, sensitivity and finally the specificity. After discussing theories, the paper will discuss the methods to create the facial recognition model. Finally, the paper will then discuss the results and analysis from the research.
Materials and methods

**Cascade Classifier**
The cascade classifier is a machine learning based method that was developed by Paula Viola and Michael Jones. The method mainly requires positive images (pictures of images we want the classifier to recognize, in this case pictures of faces) and negative images (pictures of anything that we do not want the model to recognize). The positive and negative images will first be converted into grayscale images and will be used to create a classifier that will yield a .xml file. The .xml file will then be used to detect the desired object that we want, which in this case, a face [3]. The training involves the use of 4 features as shown in Figure 1. During the training the classifier will learn the shape of faces by comparing it to the 4 features. The features can be scaled larger or smaller depending on the classifier and will move from pixel to pixel to detect the difference in color of the grayscale images in both the positive and negative images.

![Figure 1. The 4 features that are used in the training [3].](image)

**Eigenfaces**
The eigenfaces is a method of facial features detection by calculating the eigenvector and eigenvalues of faces [4]. Firstly, a picture of a face will be taken, and it will be converted into a grayscale image. The conversion is to allow more space to be saved and so that the analysis will be faster as grayscale images provides enough features to be processed and analyzed. Secondly, the matrix of the grayscale image will be calculated to find the mean face. Thirdly, the difference between the mean face and each face in the database will be calculated to produce a covariance matrix. Lastly, the eigenvectors and the eigenvalue of the covariance matrix will be calculated which will yield the eigenface which is the same as the eigenvectors.

**Raspberry Pi**
The Raspberry Pi, as shown in Figure 2, is single board computer equipped with a GPU, networking components, RAM, General Purpose Input and Output (GPIO), Bluetooth, and several ports such as USB and HDMI. The Raspberry Pi can run several operating systems such as TLXOS, Ubuntu, LibreElec, but it generally operates in Raspberry OS which was known as Raspbian. The Raspberry Pi has 40 GPIO pins that allows it to divert power to electrical components, read sensors, and run communication protocols.

![Figure 2. The Raspberry Pi [5].](image)
OpenCV
OpenCV is an open-source library focusing on Computer Vision. It was originally developed by Intel Corporation, the American tech company. Computers are unable to process images like humans, they see objects as a binary, numbers that shows the color codes, red, green, blue, and a combination of the 3 colors. OpenCV was developed in C and C++ programming language, but it has now been developed to be used in other programming languages such as MATLAB, Java, and Python.

Confusion Matrix
The machine learning models' prediction can be classified into 4 categories which are true positive, true negative, false positive, and false negative. The true category means that the model has predicted the shown value correctly or corresponding to the actual value which can be positive, in this case the model detected a face, and negative in which it did not detect a face. The false category means that the model predicted the shown value wrongly or not corresponding to the actual value which can be positive which means that the model detected a face when it was not supposed to and negative, where it did not detect a face when it was supposed to. When we combine the categories to make a table, they will form a confusion matrix that will be shown in Figure 3.

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>Positive</td>
<td>True Positive</td>
<td>False Positive</td>
</tr>
<tr>
<td>Negative</td>
<td>False Negative</td>
<td>True Negative</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. The confusion matrix table [6].

Accuracy, Sensitivity, and Specificity
The data gathered from the experiment will be analyzed with the help of 3 equations shown from Equations 1 to 3 [7]. The accuracy of the model shows the number of correct predictions that the model has made. The sensitivity of the model or SN refers to the rate of actual positive that is predicted as a positive value by the model, the sensitivity is also known as the true positive rate. The specificity of the model or SP refers to the rate where the negative value is predicted as a negative value, the specificity is also known as the true negative rate.

\[ \text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \]  \hspace{1cm} (1)
\[ SN = \frac{TP}{TP + FN} \] \hspace{1cm} (2)
\[ SP = \frac{TN}{TN + FP} \] \hspace{1cm} (3)

OpenCV Face Recognition Classifier
Before the model is able to differentiate between individual faces, it must first be able to recognize a face, what features faces have, and features in which faces do not have. For this research we will use a pre-trained classifier provided by OpenCV called the haarcascade_frontalface.xml. The classifier will be called upon in the program with the help of the cv2.CascadeClassifier as shown on Figure 4.

```python
face_detector = cv2.CascadeClassifier('opencv-4.5.2/data/haarcascades/haarcascade_frontalface_default.xml')
```

Figure 4. Part of the program that shows the algorithm to call and look for the pre-trained classifier.

Creating Datasets for Enhanced Training
The model can recognize faces after the training in the previous section [8]. However, it is still unable to differentiate between one face and another. We need to create a dataset filled with a minimum of 100 gray scaled pictures of the individual we want the machine to recognize. The first step is to create the program to take and store the pictures. The face_id function stores the id in the format of numbers, inputted manually by the user. The id later on will be associated with the face that will captured by the camera. After we input the id number, the camera will be turned on and it will take 100 photos of the person in front of the camera. The photos will then be stored in the folder dataset. Each of the photos will be named in the format "User:" + face_id + the number of photos, for example
User.1.2 is the second photo taken of the 100 total photos of the user with the id number 1. Figure 5 shows part of the program for creating the datasets for the enhanced training.

```python
face_id = input('
\n[INFO] Initializing face capture. Look the camera and wait ...
')
# Initialize individual sampling face count
count = 0
while(True):
    ret, img = cam.read()
    # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x,y),(x+w,y+h),(255,0,0), 2)
        count += 1
    # Save the captured image into the datasets folder
    cv2.imwrite(os.path.join('dataset/User.', str(face_id) + '.' + str(count) + '.jpg'),
    gray[y:y+h, x:x+w])
    cv2.imshow('image', img)
    k = cv2.waitKey(100) & 0xff
    if k == 27:
        break
    elif count >= 100:
        # Take 100 face sample and stop video
        break
```

Figure 5. Part of the program that shows the algorithm to create datasets with face ids.

**Initiating the Enhanced Training**

After the datasets have been created, we now have 100 grayscale pictures of someone’s face which we will use for the enhanced training. There are 3 unique faces used in the Enhanced training, so each of the 3 faces will have 100 grayscale images. Figure 6 shows the dataset created from running the program in Figure 5.

Figure 6. The dataset created from the program shown in Figure 5.

After the datasets have been made, we will then put the datasets to the training algorithm to initiate the enhanced training. Firstly, we must declare in the training program, the path to reach the datasets we made by filling the name of the folder in which the datasets are stored in the variable path. Secondly, we will then create the variable recognizer and fill the function cv2.face.LBPHFaceRecognizer_create() so that we can store and train the samples after we retrieve the datasets and its` ids. Thirdly, we will call the pre-trained classifier again to help the training algorithm to recognize the faces in the datasets as seen on Figure 7. Finally, we will call the getImagesAndLabels class to get the face samples with their ids accordingly in which we will finally be able to train and finally save the enhanced training in the trainer file with name and format of trainer.yml as seen on Figure 8.

```python
import cv2
import numpy as np
import os

# Path for face image database
path = 'Dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

Figure 7. Part of the training program that shows the path as well as declaring the recognizer and detector variable.
Implementing the model on the hardware

After the enhanced training had been done, the trainer.yml and the haarcascades_frontalface.xml files will be moved to the Raspberry Pi. The face recognition model will use the haarcascade_frontalface.xml file so that it can differentiate between faces and non-faces. The trainer.yml file will be used to help the model to differentiate between individual faces stored within the system as seen on Figure 9. From the previous training program, we have given each face in the datasets a unique id. The unique id will be used to help the model to assign names with our help manually. We must first initiate an id counter starting from zero. We will then create the names variable and fill it with strings with the names we want to attach to a certain id. Afterwards, we will set the camera function on the camera variable and set the size of the camera window. We will also set the size of the minimum window size to recognize a face as seen in Figure 10.

```python
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L') # grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split('.')[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,ids

recognizer.train(faces, np.array(ids))
# Save the model into trainer/trainer.yml
recognizer.save('trainer/trainer.yml')
# Print the number of faces trained and end program
print("\n[INFO] {0} faces trained. Exiting Program\n".format(len(np.unique(ids))))
```

Figure 8. Part of the training program showing the face and id gathering class and the process of the enhanced training.

Figure 9. Part of the main face recognition program that shows the use of the pre-trained and enhanced training files.

```python
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "opencv-4.5.2/data/haarcascades/haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX
```

Figure 10. Part of the main program that shows the naming of the id.

The main face recognition program has two main loops, the first loop is responsible for continuous face detection and live camera feed. The second loop will judge whether the face detected in the first loop is inside the system or not. Inside the second loop (the for loop) we will create a confidence rate that will denote how close the recognized face is to the face in the system, it will also show the name of the closest face it thinks resembles the face in the system, if it detects that the face is not the system however, it will print the word unknown as seen on Figure 11.
Figure 11. Part of the main face recognition program that shows the two main loops.

Results and discussion

Gathered Data

In order to determine the models' prediction, it is tested by showing it 100 faces for each iteration. The model will judge whether the faces shown is inside the system or not. The models' prediction will be recorded manually by the researcher with the help of the confusion matrix table which are divided mainly into 4 categories which are true positive, false positive, false negative, and true negative. The recorded data will be shown throughout Tables 1 to 5 and Table 6 will show the average of the four categories throughout the five iterations.
Table 1. Confusion Matrix of the model (1st Iteration).

<table>
<thead>
<tr>
<th>Predicted Values</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td>Positive</td>
<td>3</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Confusion Matrix of the model (2nd Iteration).

<table>
<thead>
<tr>
<th>Predicted Values</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td>Positive</td>
<td>3</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3. Confusion Matrix of the model (3rd Iteration).

<table>
<thead>
<tr>
<th>Predicted Values</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td>Positive</td>
<td>3</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Confusion Matrix of the model (4th Iteration).

<table>
<thead>
<tr>
<th>Predicted Values</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td>Positive</td>
<td>3</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5. Confusion Matrix of the model (5th Iteration).

<table>
<thead>
<tr>
<th>Predicted Values</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td>Positive</td>
<td>3</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6. Average of each category.

<table>
<thead>
<tr>
<th>Values</th>
<th>1st iteration</th>
<th>2nd iteration</th>
<th>3rd iteration</th>
<th>4th iteration</th>
<th>5th iteration</th>
<th>Average of each value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>False Positive</td>
<td>25</td>
<td>20</td>
<td>28</td>
<td>30</td>
<td>23</td>
<td>25.2</td>
</tr>
<tr>
<td>False Negative</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>True Negative</td>
<td>72</td>
<td>77</td>
<td>69</td>
<td>67</td>
<td>74</td>
<td>71.8</td>
</tr>
</tbody>
</table>

**Accuracy of the model**
The datasets used for the training were the 100 grayscale images of the 3 unique faces. The accuracy of the model shows the rate of correct predictions it has made over the total number of shown face. The scale of the accuracy is in percent. The accuracy can be calculated by adding the true positive or TP and true negative or TN then dividing them with the total value which is 100 as seen in Equation 1. The values are taken from Table 6 in which the obtained result will be 74.8%. This value indicates that the model has decent accuracy for recognizing the shape of faces as seen that the model has 0 false negatives, but it has a difficulty in differentiating between individual faces in the system as seen that there are an average of 25.2 false positives.

**Sensitivity of the model**
The sensitivity of the model shows the rate in which the actual positive value is predicted correctly as positive. The scale of the sensitivity is from 0 to 1.0, where 0 is the lowest and 1.0 is the highest. The sensitivity can be calculated by dividing the total true positive value by the sum of true positive and false negative value as seen in Equation 2. The values from Table 6 will be used to calculate the sensitivity in which the obtained value will be 1. This value indicates that the model can differentiate the 3 faces in the database accurately, but not very accurate when shown with faces outside the database.

**Specificity of the model**
The specificity of the model shows the rate in which an actual negative value is predicted correctly as a negative value. The specificity scale is in percent. The specificity can be calculated by dividing the true negative value by the sum of true negative and false positives value as seen in Equation 3. The results yielded 74.02% from using the data from Table 6. The result indicates that the model is able to differentiate faces, but it is not accurate enough to be implemented as a biometric security system as it still has a significant false positive prediction rate.

**Conclusions**
Several conclusions can be drawn from the research conducted. Firstly, the haar cascade method has a potential to make a facial recognition system despite its high error rate with the use of the pre-trained classifier. Secondly, the accuracy of the model which is 74.8% shows that the model is good at recognizing faces but has trouble differentiating between faces from the system and outside the system. Thirdly, the sensitivity of the model yields the value of 1, which means it can differentiate between the faces in the system when shown just the faces in the system. Lastly, the specificity of the model yields 74.02% which indicates that the model can differentiate between faces inside and outside of the system but not accurate enough to be implemented on a biometric security system. Future works will include making a custom classifier and an integration of an expression recognition model as well.
Conflicts of interest
The authors declare that there is no conflict of interest regarding the publication of this paper.

References