T.C.

İSTANBUL KÜLTÜR UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF

ELECTRICAL & ELCTRONICS ENGINEERING

SMART DOOR LOCK SYSTEM USING FACE RECOGNITION

Graduation Project

Name SURNAME: KEMAL CAN GÜNGÖR

2000004381

Name SURNAME: OZAN EMRE TUNCA

2000003058

Name SURNAME

Lecturer BASRİ ERDOĞAN

Supervisor

JUN 2025

ACKNOWLEDGEMENT

We would like to thank our supervisor, Lecturer Basri Erdoğan, for his invaluable guidance and support throughout this project. We would also like to thank the Department of Electrical and Electronics Engineering, Istanbul Kultur University, for providing the needed resources. Special thanks to our teachers and friends in the Electronics Laboratory for their help with data support and assistance.

ABSTRACT

This thesis presents the design and implementation of a contactless smart door lock system leveraging deep learning–based face recognition and spoof-resilient liveness detection on a Raspberry Pi 4 Model B platform. The proposed architecture begins with Multi-task Cascaded Convolutional Networks (MTCNN) for accurate face detection and alignment under varying lighting and occlusion conditions. To reduce computational load on the embedded device, HAAR-Cascade and Histogram of Gradients (HOG) extractors pre-filter candidate regions, decreasing the number of examined patches by over 50 %. For identity verification, facial embeddings are generated using InceptionResNetV1 via the FaceNet-PyTorch library; additionally, ResNet50 and VGG16 models (through Keras VGG-Face) serve as baselines in comparative transfer learning experiments on the Aberdeen dataset and a custom-collected set of 300 local subjects. A TensorFlow Lite–based liveness detector, trained on a variety of printed-photo and replay-video spoof examples, provides robust protection against presentation attacks in everyday use. Integrated into a Raspberry Pi 4–based face verification pipeline, the system maintains reliable recognition performance at nearly one frame per second, all without relying on any additional hardware accelerators.

ÖZET

Bu tez, Raspberry Pi 4 Model B platformunda derin öğrenme tabanlı yüz tanıma ve sahteciliğe dayanıklı canlılık tespitinden yararlanan temassız akıllı kapı kilidi sisteminin tasarımını ve uygulamasını sunmaktadır. Önerilen mimari, değişen aydınlatma ve tıkanıklık koşulları altında doğru yüz tespiti ve hizalaması için MTCNN ile başlar. Gömülü cihaz üzerindeki hesaplama yükünü azaltmak için HAAR-Cascade ve HOG çıkarıcılar aday bölgeleri önceden filtreleyerek incelenen yamaların sayısını %50'den fazla azaltır. Kimlik doğrulaması için, yüz yerleştirmeleri FaceNet-PyTorch kütüphanesi aracılığıyla InceptionResNetV1 kullanılarak üretilir; ayrıca, ResNet50 ve VGG16 modelleri (Keras VGG-Face aracılığıyla) Aberdeen veri kümesi ve 300 yerel denekten oluşan özel olarak toplanmış bir küme üzerinde karşılaştırmalı transfer öğrenme deneylerinde temel olarak hizmet eder. Çeşitli basılı fotoğraf ve tekrar oynatma videosu sahte örnekleri üzerinde eğitilen TensorFlow Lite tabanlı bir canlılık dedektörü, günlük kullanımda sunum saldırılarına karşı sağlam koruma sağlar. Raspberry Pi 4 tabanlı bir yüz doğrulama hattına entegre edilen sistem, herhangi bir ek donanım hızlandırıcısına güvenmeden saniyede neredeyse bir kare hızında güvenilir tanıma performansını korur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

V Voltage

A Ampere

GPIO General Purpose Input/Output

CPU Central Processing Unit

GPU Graphics Processing Unit

MOSFET Metal Oxide Semiconductor Field Effect Transistor

Ω Ohm

Com Common

TTL Transistor-Transistor Logic

$\mathbf{R}_{\mathbf{DS(on)}}$ drain-source open resistance

IR Infrared

HSM Hardware Security Module

FPS Frame Per Second

1. INTRODUCTION

Today, traditional lock systems are becoming increasingly inadequate in protecting individuals, institutions and especially areas requiring high security. In response to this need, biometric recognition technologies have come to the fore and become an important alternative in the field of security. In particular, facial recognition systems surpass other biometric methods with their contactless operation, user-friendly structure and high accuracy rates. With the rapid developments in artificial intelligence technologies, the areas of use of facial recognition systems have also expanded. It is now widely preferred not only in physical security; but also in many different areas such as digital identity verification, access control and human-machine interaction.

1.1. Technological Motivation and Platform Selection

The fact that technology giants such as Apple, Microsoft and Google have integrated facial recognition systems into their products is one of the clearest indicators of how important this technology has become. However, a large part of the existing systems require devices with high hardware power and are generally built on closed-source software. In this context, in this study, a facial recognition-based smart door lock system supported by deep learning algorithms has been developed running on Raspberry Pi 4, a low-cost and open-source platform. Raspberry Pi; It offers a very suitable infrastructure for security applications with its small size, low power consumption, hardware control thanks to GPIO pins, and compatibility with common programming languages such as Python. In addition, it can be easily adapted to different usage scenarios thanks to its portability and flexible hardware integration. The system detects people and authenticates them through live images taken through the camera. One of the most striking aspects is that it can work without having to load user data into the system in advance. When it detects a person for the first time, it extracts their face vector (embedding) and saves it in the system, and in subsequent encounters, it recognizes this person and grants access. Thus, the system improves itself over time and becomes a more efficient security solution.

1.2. Facial Data Security and Algorithmic Framework

In face recognition systems, the processing, storage and secure management of personal data are of great importance both ethically and legally. The system developed in this direction processes and stores user data only locally and does not transfer it to any external server. Thanks to this structure, users' facial information is protected against external threats and data security is provided at a high level. In addition, the system only stores the basic face data required for recognition and uses this data only during the relevant process. This approach also offers a significant advantage in terms of compliance with legal regulations such as the Personal Data Protection Law (PDPL). In the face detection part of the system, the Multi-Task Cascaded Convolutional Neural Networks (MTCNN) algorithm, which is based on deep learning and has a multi-task structure, was preferred. MTCNN not only detects the position of the face, but also determines basic facial points such as eyes, nose and mouth and performs face alignment. In this way, more regular and normalized data is provided to the face recognition model, and the accuracy of the recognition process is increased.

In the face recognition phase, the InceptionResNetV1 model, which is a combination of the Inception architecture developed by Google and the Residual Network (ResNet) architecture developed by Microsoft, was used. This model can successfully analyze facial details at different scales with its multi-filter structure; at the same time, it minimizes data losses that may occur when working with deep layers thanks to residual connections.

1.3. Literature Review

Facial recognition technology has become one of the most important research topics in the fields of computer vision and artificial intelligence today. Thanks to its potential to automate identity verification processes, it has rapidly become widespread as one of the basic components of security systems. Developed facial recognition systems are actively used in many different application areas such as security cameras, mobile devices, access control points and biometric access systems. Commercial applications carried out by major technology companies have been effective in the spread of this technology in daily life. For example, Apple has introduced a significant innovation by integrating facial recognition into mobile devices with the Face ID technology it developed with the iPhone X model it introduced in 2017 (Apple Inc., 2017). This system uses depth data by creating a threedimensional map of the face and stores the user's biometric information only within the device. Apple also provides confidence in data privacy by emphasizing that user data is never transferred to external servers. Similarly, Microsoft has integrated facial recognition technology into computers via the Windows Hello platform it offers with Windows 10 (Microsoft Corp., 2016). Thanks to infrared cameras and artificial intelligence-supported recognition algorithms, users can securely access their devices without entering a password. This system is also widely used at the corporate level, supported by alternative authentication methods such as fingerprint and PIN. Google supports facial recognition technology via Android operating systems. Especially in Android 10 and later versions, facial recognition functionality has been integrated into the systems via biometric APIs (Google LLC, 2020). These solutions offered by Google combine camera-based facial detection processes with deep learning algorithms, providing user verification processes that are both fast and adaptable to different security levels.

1.4. Comparative Analysis of Face Detection and Recognition Techniques

Among the face recognition models based on deep learning architectures, VGG16, ResNet50, and InceptionResNetV1 stand out. Smith et al. (2022) tested the VGG16 model on LFW and CelebA datasets, achieving an accuracy rate of approximately 95% and demonstrating the face recognition performance of this model. Muthu et al. (2023) emphasized the robust structure of the model by reporting a 98% accuracy rate on the same datasets in their study with the ResNet50 architecture. Similarly, Jones et al. (2023) achieved 98.5% accuracy using the InceptionResNetV1 model, which is a combination of Inception and ResNet architectures. These results show that deep learning-based architectures offer high success in face recognition tasks. In face detection, historically important classical methods — HAAR-Cascade and HOG — have fallen behind modern deep learning-based approaches today. Lee et al. (2021) reported an accuracy rate of 88% in their study using the HAAR algorithm. Chen et al. (2022) stated that they achieved an accuracy rate of 91% by testing the HOG algorithm. Compared to these traditional methods, the MTCNN (Multi-Task Cascaded Convolutional Neural Networks) algorithm provides higher success in both face detection and detection of key points (landmarks) of the face. Wu et al. (2023) reported that the MTCNN algorithm worked with an accuracy rate of 94% and proved its effectiveness in face alignment tasks. Face recognition solutions integrated with embedded systems are preferred especially in small-scale applications thanks to their advantages such as portability, low energy consumption and cost-effectiveness. Razzaque et al. (2021) emphasized that the Raspberry Pi platform is widely used in IoT, smart home systems and access control devices. In the literature, there are also sample applications where face recognition systems are integrated into smart door lock mechanisms. Shamrat et al. (2021) developed a face recognition system working with a HAAR classifier and integrated this system into the door lock mechanism. However, the accuracy rates of such systems remain lower compared to deep learning-based models.

Table 1. Literature comparison table

Authors	Method/Model	Dataset	Accuracy
Lee, S. et al. (2021)	HAAR-Cascade	WIDER FACE,LFW	88%
Chen, L. et al. (2022)	HOG	WIDER FACE,LFW	91%
Wu, Y. et al. (2023)	MTCN	WIDER FACE,CelebA	94%
Smith, J. et al. (2022)	VGG16	LFW,CelebA	95%
Muthu, K. et al. (2023)	ResNet50	LFW,CelebA	98%
Jones, A, et al.(2023)	IncepitonResNetV1	LFW,CelebA	98.5%

2. MATERIAL AND METHODS

An extensive account is presented here of the algorithmic techniques and hardware architecture used to create the facial recognition-based smart door lock system, designed to provide good security by consuming fewer resources. The design chase of the system is to provide real-time operation capability for a low-cost single-board computer: Raspberry Pi 4 Model B. Functions of key hardware components, such as solenoid door lock, a 5V relay module, and Pi Camera V2 for image capture and access control, are described. On the software side, the present system uses the facenet framework to combine traditional computer vision methods with modern deep learning-based models like MTCNN, VGG16, ResNet50, and InceptionResNetV1. The facial recognition pipeline consists of face detection, alignment, embedding extraction, and verification of identification by cosine similarity as the distance metric.

2.1. Hardware Used

The hardware structure is configured to support functions such as facial image acquisition, processing, and physical door control. Raspberry Pi 4 Model B was chosen as the processing unit at the center of the system, especially thanks to its low power consumption, compact structure, and wide input-output support. The camera and control elements integrated into this hardware contribute to the holistic operation of the system.

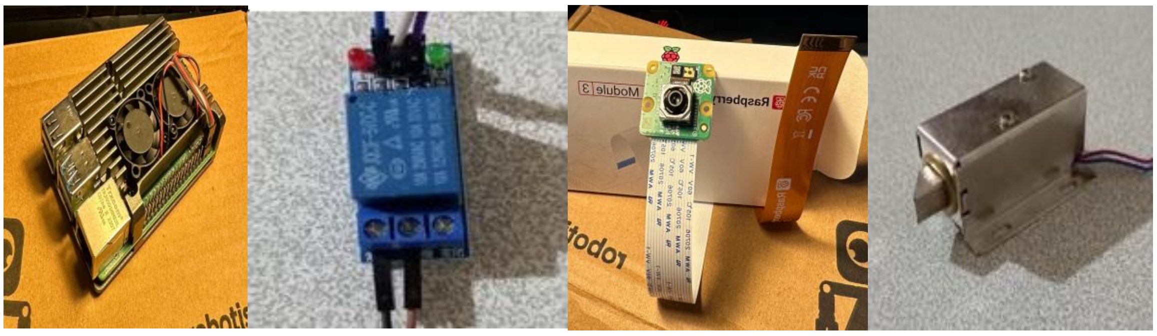

Figure 1. Hardware components: (a) Raspberry pi 4 model b, (b) 5V relay module (c) Pi camera V3, (d) Solenoid door lock,

Raspberry Pi 4 Model B

Raspberry Pi 4 serves as the central processing unit of this system. Both image processing and physical control operations are performed on this device. Raspberry Pi, which is widely preferred in embedded systems, offers a platform suitable for security applications with its low power consumption and wide peripheral support.

Technical Specifications:

• 1.5 GHz quad-core ARM Cortex-A72 processor (64-bit)

• 4 GB LPDDR4 RAM

• 2 micro-HDMI outputs (4K supported)

• 2 USB 3.0 and 2 USB 2.0 ports

• Built-in Wi-Fi 802.11ac and Bluetooth 5.0

• 40-pin GPIO (General Purpose Input Output) interface

• MicroSD card input (for operating system and data storage)

• CSI (Camera Serial Interface) and DSI (Display Serial Interface) connectors

• 5V/3A power input (via USB-C port)

Pi Camera Module V3

This module, used to obtain facial images, is a high-resolution camera unit that can be directly connected to Raspberry Pi. The images are processed in real time by the system and transmitted to the facial recognition algorithms.

Camera Information:

• Sony IMX219 sensor, 8 megapixel resolution

• Can be connected directly to Raspberry Pi via CSI port

5V Relay Module

Used to control electrical loads (such as solenoid locks) that operate with higher voltage and current using digital signals generated by Raspberry Pi. When a user recognized by the system is detected, this module is triggered and the door lock is opened.

Solenoid Door Lock

It is an electromechanical locking system that allows the door to be physically opened and closed. When triggered by the relay, it receives energy and releases the lock mechanism for a short time, allowing the door to open. It is widely used in security systems.

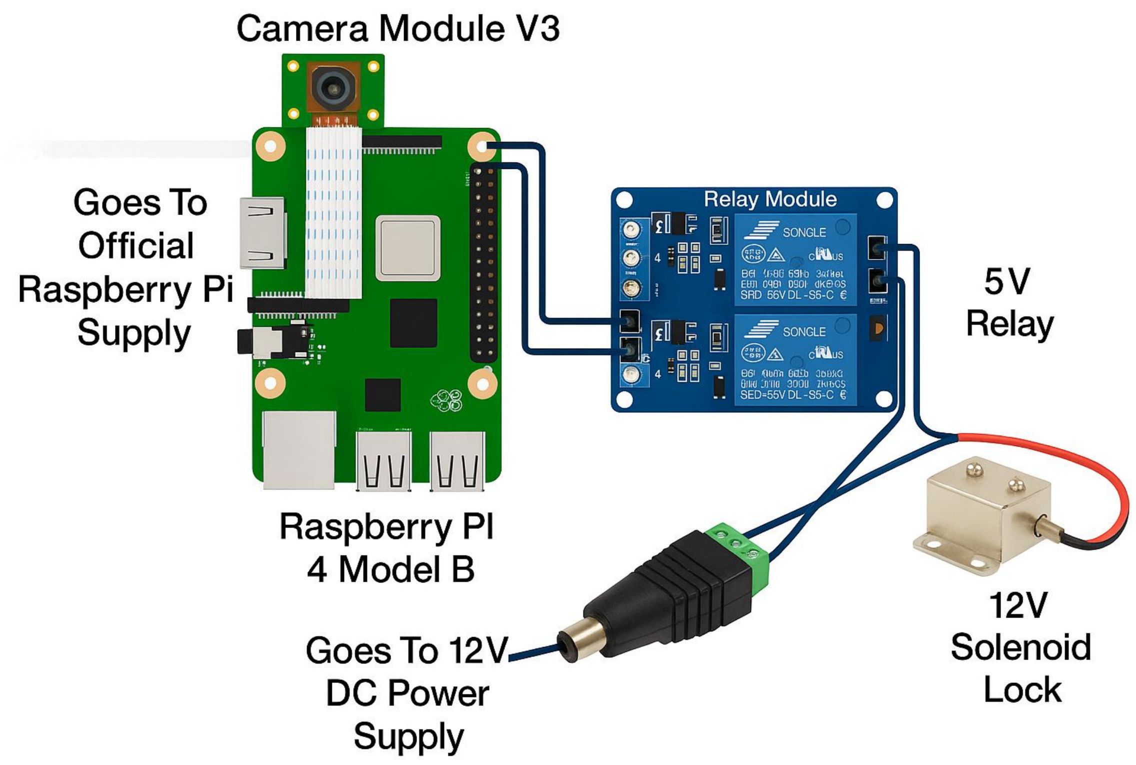

Figure 2. Design of the study

2.2. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source software library created for the purpose of developing image processing and computer vision applications. This project, which was first launched by Intel in 2000, can today work on many different platforms and programming languages. It is widely used, especially with the Python language. OpenCV offers a wide range of tools for both simple and advanced image processing operations.

In the face recognition based smart door lock system developed within the scope of this project, OpenCV has formed the basic image processing infrastructure. Functions such as receiving images from the camera module, preparing each frame for processing and providing visual feedback to the user were performed with OpenCV. First, the system was provided with real-time image acquisition, then these images were subjected to color space transformation to make them suitable for the MTCNN model. After the face detection was completed, the recognition process was performed; the name of the recognized person was written on the image, and the word “unknown” was added for unregistered persons. In addition, images belonging to unrecognized persons were automatically recorded by the system.

2.3. FaceNet

FaceNet is a deep learning architecture developed by Google and considered a significant milestone in the field of face recognition. This model, presented by Schroff, Kalenichenko and Philbin in 2015, aims to obtain a face embedding vector directly from a face image. Thanks to these vectors, faces belonging to the same person are represented in a numerical space in a way that they are close to each other and faces belonging to different people are far away. One of the most striking features of FaceNet is that it can perform face recognition, verification and clustering operations without the need for a classical classification approach. During the training of the model, a specially designed triplet loss function is used instead of traditional cross-entropy. This loss function works on triple samples, including a reference face (anchor), another face belonging to the same person (positive) and a face belonging to a different person (negative). Thanks to this structure, the model learns the differences between faces more clearly and maximizes its ability to distinguish.

2.4. VGG-Face

VGG-Face is a deep learning-based model developed by the Visual Geometry Group (VGG) at Oxford University and trained specifically for face recognition applications. This model, based on the VGG architecture, has been trained on large-scale face datasets and achieved high accuracy rates. The model, which is based primarily on the VGG16 architecture, has a deep and regular structure consisting of 13 convolutional layers and 3 fully connected layers. VGG-Face can be used not only in face recognition but also in operations such as face verification and face clustering. One of the most important advantages of VGG-Face is its high depth and learning capacity. In this way, the model can learn more complex facial features and produce more precise results.

2.5. Deep Learning Models

In this section, the basic deep learning models used in our system will be introduced; the architectural structures and functionalities of the preferred architectures in face detection, feature extraction and liveness verification stages will be briefly summarized.

2.5.1. InceptionResNetV1

InceptionResNetV1 is a hybrid deep learning model consisting of the Inception architecture developed by Google and the Residual Network (ResNet) structure developed by Microsoft. This model is designed to provide both multi-scale feature extraction and increased depth so that deep neural networks can work more effectively in tasks such as classification and recognition. It stands out especially in the field of face recognition with its high success. Combining the strengths of Inception and ResNet architectures, this structure allows the model to learn more deeply and increases the generalization capacity of the features. Especially when used with the FaceNet architecture, it performs the embedding process by converting a face image into a multi-dimensional (e.g. 512-dimensional) digital vector. This vector is compared with the ones previously recorded in the system and the person's identity is determined.

2.5.2. ResNet50

ResNet50 is a 50-layer version of the Residual Network (ResNet) family developed to overcome learning difficulties encountered in deep convolutional neural network (CNN) architectures. Introduced by Microsoft Research in 2015, this structure uses a special layer structure called residual connections to solve problems such as gradient loss and over-learning seen in deep networks. Thanks to these connections, the input data can be directly transferred to the next layers and the learning process becomes more stable. The most important feature of the ResNet architecture is that it allows the model to have deeper layers with the "shortcut" connections in each block, while achieving high accuracy without complicating the training process.

2.5.3. VGG16

VGG16 is a deep convolutional neural network (CNN) architecture developed by the Visual Geometry Group (VGG) affiliated with the University of Oxford and is frequently used in tasks such as image classification and object recognition. The model attracted attention with its success in the ImageNet competition held in 2014 and has been widely adopted. The “16” in its name represents 16 weighted layers consisting of a total of 13 convolutional and 3 fully connected layers. In face recognition applications, a special version of this architecture called VGG-Face is usually used. VGG16 has a very deep and complex structure. This causes the model to require powerful hardware requirements and long training times. Nevertheless, it is frequently preferred because its deep structure has a high capacity to increase classification accuracy. While its advantages include rich feature extraction and high success when trained with large data sets; its disadvantages are that the training process is time-consuming and the computational cost is high due to the large structure of the model.

2.6. ATTRIBUTES

In this section, the hardware and software features of the basic components of our system will be detailed. Critical criteria such as the number of layers of learning models, parameter sizes, execution times and memory requirements will be discussed under this subheading.

2.6.1. HAAR CASCADE

These are features developed by Viola and Jones in 2001 that allow for the rapid detection of specific patterns in an image (edges, corners, flat areas, etc.). These features allow statistical analysis of specific structural differences in specific objects, such as a human face (for example, darker eye contours, lighter forehead and cheeks). Haar features are based on pixel intensity differences between rectangular regions placed at different sizes and positions. These differences are evaluated by a classifier trained on positive and negative examples. The algorithm uses a large number of such rectangular features to quickly scan objects with fixed structures, such as faces, eyes, and mouths.

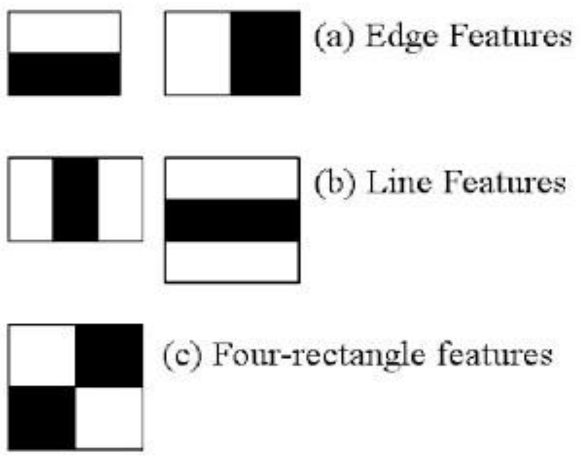

Figure 3. HAAR features; (a) edge, (b) line and (c) 4-rectangle features.

In addition, the Haar Cascade method allows more detailed analysis to be performed only in possible regions by passing each image region through a series of progressive classifiers instead of directly analyzing each region in detail. This structure creates the mechanism called “cascade”, i.e. progressive evaluation.

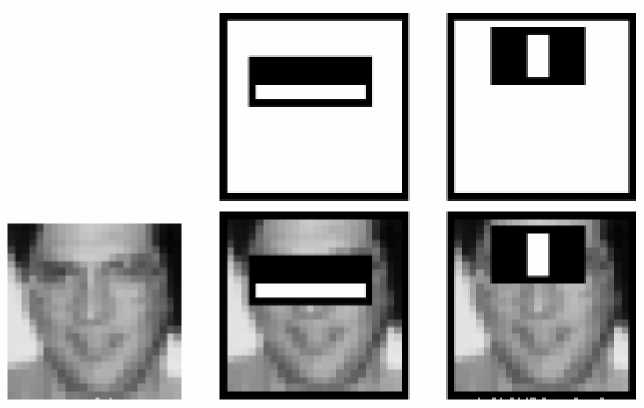

Figure 4. How HAAR features are applied to the human face.

```
let faceCascade = new cv.CascadeClassifier();
let eyeCascade = new cv.CascadeClassifier();
faceCascade.load('haarcascade_frontalface_default.xml');
eyeCascade.load('haarcascade_eye.xml');
```

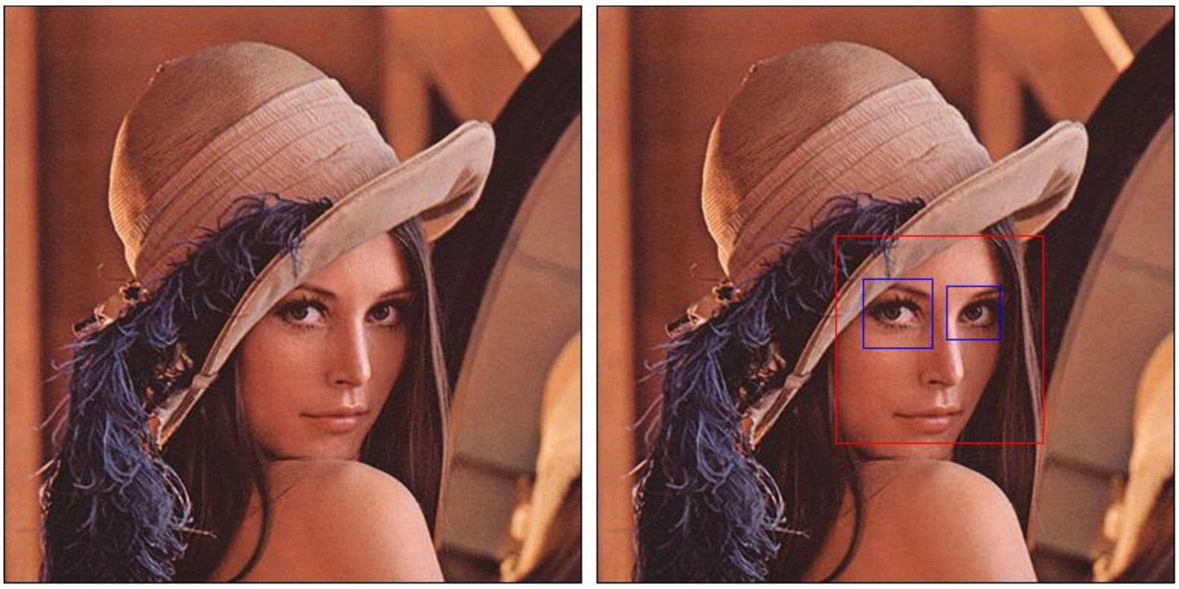

Figure 5. How Haar Cascade captures the human face.

2.6.2. MTCNN

MTCNN (Multi-task Cascaded Convolutional Neural Network) is a multi-task deep learning model that can simultaneously perform the tasks of face detection and detection of structural points (landmarks) on the face. (Zhang et al,2016). MTCNN, which goes beyond classical methods and offers more flexible and accurate face detection, can work with high success rates, especially under variable face positions, facial expressions and lighting conditions. One of the strongest aspects of the model is that it not only determines the position of the face, but also provides the structural points required for the face alignment process. (Wu et al,2023) In this way, the face can be aligned correctly even when viewed from different angles. A correctly aligned face during the recognition phase ensures that the embedding inference is more reliable and reliable. MTCNN has a three-stage cascade neural network structure:

P-Net (Proposal Network): Roughly estimates the regions in the image where there may be a face.

R-Net (Refine Network): Increases accuracy by eliminating these regions.

O-Net (Output Network): Refines facial positions and also detects landmarks (two eyes, nose, corners of the mouth).

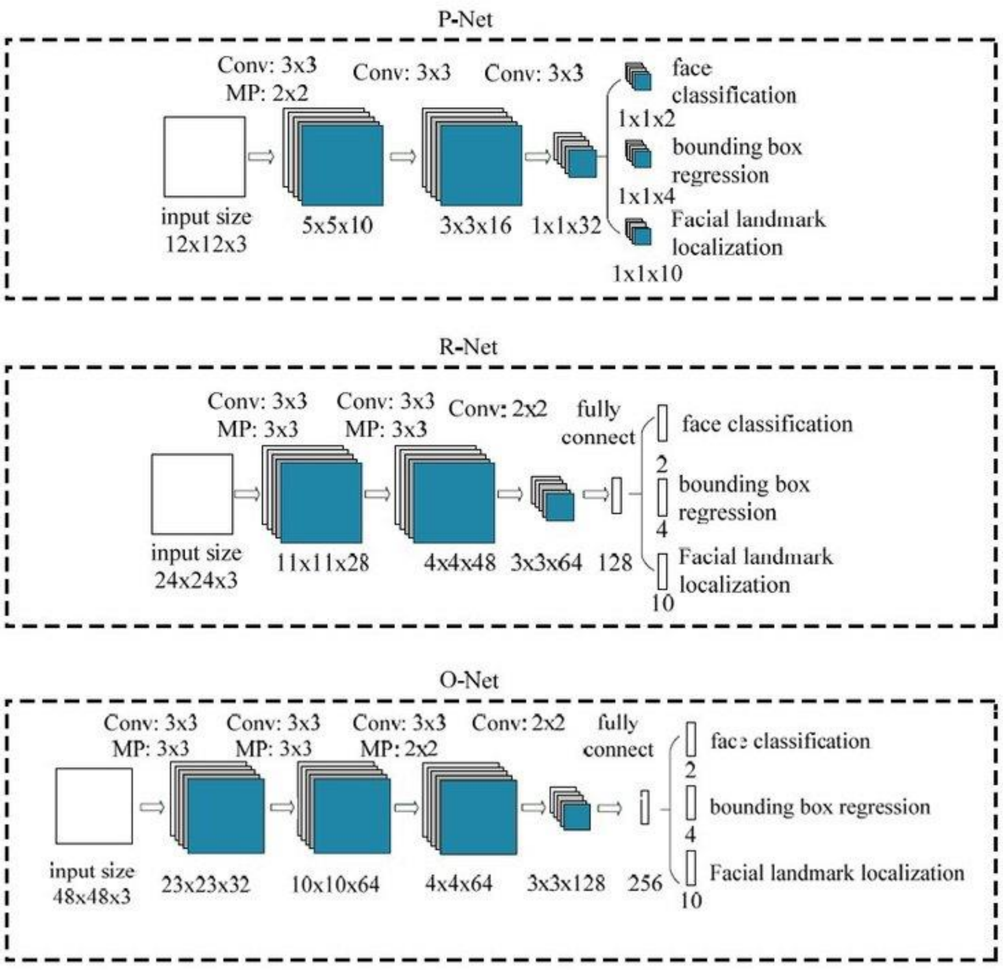

Figure 6. MTCNN stages

```
detector = MTCNN()
img = cv2.imread("img.jpg")
detections = detector.detect_faces(img)
for detection in detections:
    score = detection["confidence"]
    if score &amp;amp;gt; 0.90:
        x, y, w, h = detection["box"]
        detected_face = img[int(y):int(y+h), int(x):int(x+w)]
```

2.6.3. HOG (Histogram of Oriented Gradients)

HOG (Histogram of Oriented Gradients) is a classical feature extraction method used to identify objects or structural elements in an image. (Dalal & Triggs, 2005) First proposed by Dalal and Triggs in 2005 for human detection, this method uses edge information and gradient orientations in the image to represent the distinguishing features of objects in a vector format. The use of HOG in tasks such as face recognition and detection allows for statistical modeling of structural and shape-based information in the image. However, the main limitation of this method is that it relies only on low-level edge information and cannot learn information that carries deep contextual meaning. Therefore, HOG may show lower accuracy compared to deep learning-based approaches in situations such as variable pose, expression, partial occlusion, and low light.

The basic principle of the HOG algorithm is to divide an image into small cells and calculate the gradient (edge) directions of the pixels in each cell. These orientations are then represented as a histogram. The histograms are then normalized at the block level to provide robustness against changing lighting conditions.

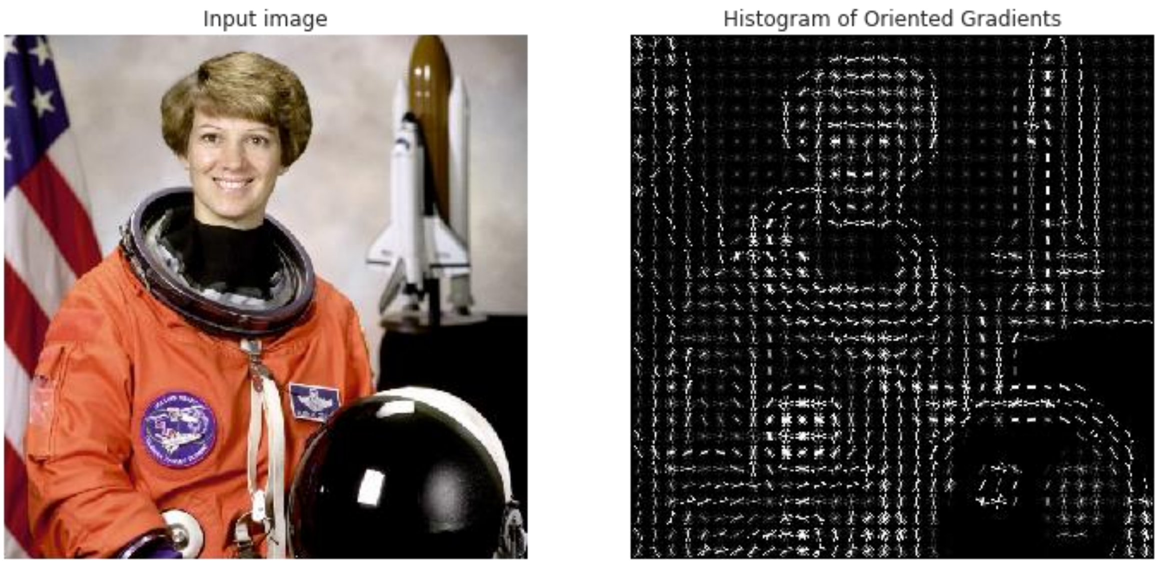

Figure 7. (1) Input image, (2) HOG feature descriptor.

2.7. Auxiliary Algorithms

In this subsection, additional algorithms that support the basic functions of the system will be examined; the purposes, steps and integration forms of auxiliary techniques such as face region preprocessing, histogram equalization, image filtering and similar will be explained.

2.7.1. Cosine Similarity

Cosine similarity is a method widely used especially in fields such as text classification and information retrieval. It is used to measure the degree of similarity between two vectors and takes values between 0 and 1. A cosine similarity of 0 indicates that the two vectors are completely different, while a cosine similarity of 1 indicates that they are completely the same. The FaceNet-based InceptionResNetV1 model used in this project represents each face image as a multidimensional embedding vector. Cosine similarity was used to measure how similar these vectors are to each other. The estimated result was determined as 0 or 1 by applying a threshold value to the result obtained from the Cosine similarity formula given in Equation 1.

$$S_C(A,B) = \cos(\theta) = \frac{A \cdot B}{\| A \| \, \| B \|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

2.8. The Parameters of Optimizations

This subsection will discuss the main hyperparameters and optimization strategies tuned to optimize the overall performance of deep learning models and the system. In addition to factors such as learning rate, batch size, and epoch number in model training, we will also detail how to determine the face recognition threshold, liveness test threshold, and image preprocessing settings.

2.8.1. Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is the most primitive of optimization algorithms, which updates the model weights with the gradient obtained using a mini-batch of randomly sampled elements instead of the whole set. This gives low memory consumption and helps to get out of local minima of the cost function. However, the same gives rise to slow convergence sometimes as it works according to a fixed learning rate. Therefore, it is mostly extended with the momentum term. Momentum adds stability in the update of weights regarding the gradient information from the previous steps.

2.8.2. Adam (Adaptive Moment Estimation)

Adam is an algorithm that updates model parameters using moving averages of both gradients and gradient squares. Generally, this method would provide a rapid and efficient solution but may demand varied learning rates based on model size and complexity. However, due to the dynamic characteristic of Adam, it performs consistently on various datasets.

2.8.3. RMSProp (Root Mean Square Propagation)

RMSProp was created to counteract the decline of Adagrad's learning rate. By using a weighted moving average of the gradient squares, it keeps the learning rate constant while generally providing better stability. As such, it often comes with the need for additional hyper-parameters to be fine-tuned, which can easily complicate the model.

2.9. Basic Operation of the Project

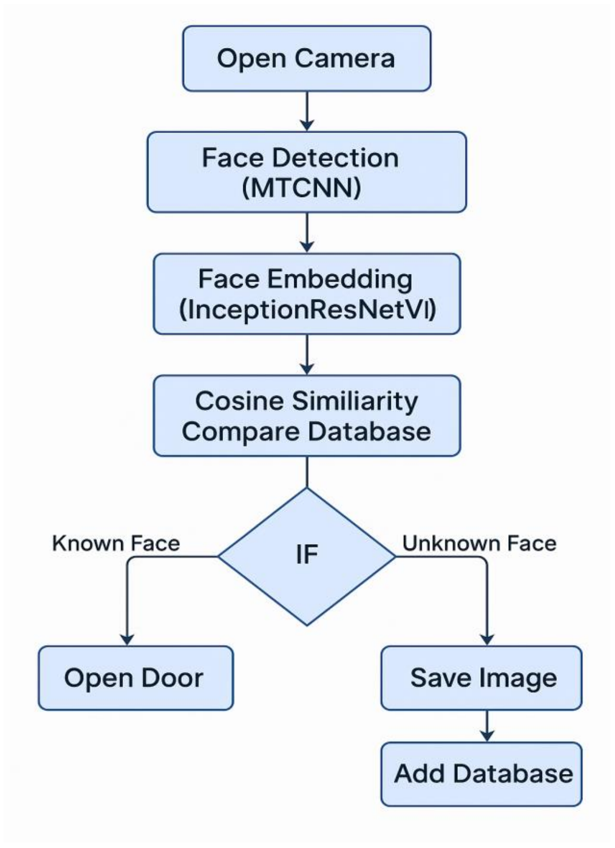

Figure 8. Workflow diagram of the study

First, camera opens, and detects a person's face. The detected facial image is recognized using a trained model. The estimated results are evaluated using a cosine similarity method. Recognition success of the estimated face will be given a percentage success score. If the estimate is recognized, the lock opens; if not, it stays shut. The system continues the face detection process until the user exits the program.

2.10. Dataset

In this study, the Aberdeen psychological dataset from the Psychological Image Collection (PICS) at the University of Stirling was used to improve the performance of the system with regard to face recognition, to improve its generalization capability against general facial structure, and finally to increase the success rate against different facial features. (Muthu et al., 2023) The Aberdeen dataset comprises 687 color face images from a total of 90 individuals. However, for the purpose of this study, it targeted a person only with sufficient data to be able to train the model well. That is why persons who had less than 18 images were filtered out from the dataset and hence 23 persons having 18 images or more were chosen. Taking 18 images from each person, the resultant new sub-dataset comprised a total of 414 face images. In training, 75% of the data were used as train data, and the rest 25%-test data. In this regard, 310 face images were used for training while 104 face images were used for testing. The images are in RGB format and determined input dimensions are the pixels of 224x224. During model training, batch size value was selected as 16 considering hardware capacity of the system. A low batch size value slows down the training process, whereas higher batch size values risk going beyond the capacity of the memory and processor resources of the Raspberry Pi device.

Figure 9. Example human face photographs from the dataset

2.10.1. Usage of Dataset in Training Process

At the initial training sessions, training was done using the Keras VGG-Face model. But the major drawback of this model is that it requires retraining of the model from scratch when a new user needs to be added to the model after the completion of the training. This is a serious disadvantage especially for systems that are working in real-time and can keep adding users continuously. The above abstains from this problem and enables the system to be much more flexible by the pre-trained InceptionResNetV1 model in the FaceNet-PyTorch library. The architecture FaceNet converts the users face images to embedding(vector) and keep these in the system. On adding a new person, embedding data of that person alone get in and complete re-training of the model is not required. The system, therefore, provides both high accuracy and the possibility of easily and quickly adding new users to the system. Moreover, such an approach made the most efficient use of the system resources especially on limited resource hardware platforms such as Raspberry Pi and thus, made a excellent advantage for real-time applications.

2.11. Evaluation Metric

The metric used to evaluate the performance of the study provides Accuracy values for each method combination. The accuracy value is calculated by the following Equation 2:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

In this equation, TP (True Positive) is represented as correctly predicted positive values, TN (True Negative) as correctly predicted negative values, FP (False Positive) as incorrectly predicted positive values, and FN (False Negative) as incorrectly predicted negative values.

2.12. Power Management and 3.3 V–5 V MOSFET Based Driver Circuit

One of the main purposes of this project is to drive a 5V relay and solenoid door lock with the GPIO pins (3.3V logic output) of the Raspberry Pi 4 Model B without any problems. Since the GPIO pins of the Raspberry Pi can only go up to 3.3V, connecting these pins directly to a 5V relay coil will not only not provide enough voltage, but will also strain the internal protection circuits of your Raspberry Pi and cause stability problems. For this reason, a MOSFET switching circuit has been designed that converts the 3.3V logic output to 5V logic level and can also transfer an external 5V supply to the relay coil.

2.12.1. Circuit Working Principle

1. Initial State (GPIO LOW)

1. Raspberry Pi GPIO pin is in 0 V (LOW) state.

2. MOSFET's Gate end is not pulled to 3.3 V via R1; since the voltage between gate and source is zero, MOSFET is cut off.

3. Meanwhile, Drain end is pulled to +5 V level via R2 pull-up (10 kΩ). In other words, the "logic signal" going from MOSFET to relay logic input is 5 V (HIGH).

4. If your relay module is a "LOW activating" type, in this case, when 5 V logic comes to the relay circuit, the relay coil is not energized and the door lock remains closed.

2. Active State (GPIO HIGH = 3.3 V)

1. Raspberry Pi GPIO pin goes to 3.3 V (HIGH).

2. Gate end goes to 3.3 V level via R1 and exceeds the gate-source threshold voltage, thus making MOSFET conductive.

3. When the MOSFET is turned on, the resistance between Drain (D) and Source (S) drops to a very small value and the Drain end is connected to the ground (GND of the Raspberry Pi). Thus, the Drain output of the MOSFET goes to the ground (LO G = 0 V).

4. At this point, the “5 V logic output (the point that goes to the relay logic input)” goes to the LOW level (approximately 0 V), i.e. it activates the relay module. The relay coil is energized from the 5 V power supply line inside and the door is opened by driving the solenoid door lock.

2.12.2. Advantages of BS170 MOSFET

Logic-Level Feature: BS170 is a “logic-level” MOSFET with a gate-source threshold of approximately 2–3 V. Raspberry Pi’s 3.3 V output is sufficient to trigger the BS170’s gate without any problems.

Convenient Package and Easy Installation: BS170’s TO-92 package size allows easy integration into breadboards and small PCBs.

Low Channel Resistance (R_DS(on)): Thanks to its sufficiently low conduction resistance (~5 Ω), when the MOSFET is conducting, it quickly drops the drain end to GND potential and feeds the relay coil with full current.

3. IMPLEMENTATION AND RESULTS

In this section, the implementation parts of the hardware and software components of our smart door lock system project and comprehensive results are presented. First, the general structure and working steps of the system will be explained, then real-time face recognition analysis, face addition and will be explained in detail. Finally, measurements and evaluations of system performance are included.

3.1. Designed System

The developed smart door lock system integrates advanced facial recognition and liveness detection techniques to provide secure and user-friendly access control. The main application script, raspberry_gui.py, manages real-time video acquisition, face detection, embedding computation, and lock actuation. A dynamic preprocessing pipeline including automatic brightness adjustment and adaptive thresholding ensures robust performance under varying lighting conditions. The GUI employs multithreading to maintain a responsive interface during user registration and verification, preventing freezes and lag.

Key functional modules:

1. **Image Acquisition & Preprocessing**: Continuous capture from Raspberry Pi Camera V3; normalization, face region cropping, and noise reduction.

2. **Face Detection & Alignment**: MTCNN detects faces and facial landmarks to align and crop regions for consistent embedding extraction.

3. **Feature Extraction**: InceptionResNetV1 generates 512-dimensional embedding vectors for each detected face.

4. **Liveness Verification**: A TensorFlow Lite anti-spoofing model evaluates each face crop to prevent presentation attacks.

5. **Embedding Comparison**: Cosine similarity compares live embeddings against stored profiles, using a configurable threshold for identity verification.

6. **Access Actuation:** GPIO-driven 5V relay triggers the solenoid lock when similarity exceeds the threshold, granting access.

7. **Logging & Feedback**: Recognition events and system actions are recorded in actions.log, while the GUI presents real-time status updates.

These components work in concert to achieve high recognition accuracy and maintain low latency, making the system suitable for both industrial and educational environments.

3.2. System Architecture and Flow Diagram

This section will see the hardware and software layers of the system and the data and control flow between them.

3.2.1. Hardware Architecture

Table 2. Detail of components

Component	Explanation	Connection Details
Raspberry Pi 4 Model B	Main control unit; all software processes run here.	GPIO17 output is connected to the relay, camera CSI port is connected to Pi Camera.
Pi Camera Module V3	Camera module providing video streaming at 640×480 resolution.	It is connected to the Raspberry Pi's special camera port via the CSI interface.
5V Single Channel Relay Module	The solenoid acts as an electronic switch to open and close the lock.	Relay input pin (IN) to Raspberry Pi GPIO17, VCC-5V, GND-GND.
Solenoid Door Lock	Mechanical lock unit that opens and closes with an electrical signal.	Connected to the NO/COM terminals of the relay; normally 5 V adapter power.
5V/3A Adapter	Provides constant power for relay/solenoid.	The adapter’s 5V output is connected to the Pi USB-C.

GPIO Cable and Splitter	It allows easy connection of GPIO pins on Raspberry Pi.	GPIO17, GND, 5V pins are connected to the relay module.

Hardware Flow:

1.Camera Flow: Pi Camera Module V3 produces minute resolution raw data via CSI port.

2. Processing Unit: Raspberry Pi 4 processes incoming frames in software and performs face detection, liveness control and embedding extraction.

3. Relay Trigger: In case of access approval during the verification phase, the relay is triggered via GPIO17.

4. Solenoid Lock Management: When the relay is active, the solenoid door lock is opened; when it is passive, the lock remains closed.

3.2.2. Software Architecture

1. Image Capture and Preprocessing

Raw frames coming from Picamera2 are converted to grayscale, followed by histogram equalization. It assures proper detection of faces even in less luminous conditions.

2. Face Detection and Alignment (MTCNN)

The MTCNN model performs face detection and localizes the five basic landmarks (eyes, nose, mouth corners) within each frame. The face region is aligned and cropped to 160×160 pixels based on the landmark locations.

3. Liveness Check (TFLite-Anti-Spoofing)

The face crop is processed with an anti-spoofing model based on TFLite. If an identity verification system operates with the fake entries at less than 0.5, then photos and videos shall be rejected.

4. Embedding Inference (InceptionResNetV1)

The face crop that passes the liveness test is transformed by the InceptionResNetV1 into 512-dimensional feature vectors known as embeddings.

5. Comparison and Authentication

The new embedding is compared to the profile embeddings stored in the database using torch.cosine_similarity. A similarity score of 0.9 or greater is accepted as a match; anything lower is deemed a rejection.

6. Decision on Access and Relay Triggering

A match triggers the relay with GPIO.output(RELAY_PIN, GPIO.HIGH), opening the solenoid lock. In the no-match scenario, the relay stands still, and the user encounters the message "Access Denied".

7. Logging and Feedback

Entries with timestamps will be created in actions.log upon every recording or verification performed through the App.log_action() method. In the GUI, the user is notified of the transaction status with short texts such as “Face detected”, “Liveness successful” or “Verification Failed”.

3.2.3. Control and Data Flow Diagram

When an application starts, the system starts working by initializing the camera interface and GPIO pins on the Raspberry. In every new frame of the live video loop, it is processed with grayscale and histogram equalization to maximize contrast. Then, the MTCNN model detects faces and five basic landmarks on the frame. Landmark coordinates serve as references for alignment and cropping of the face area.

Image Preprocessing: Grayscale and histogram equalization allow clear detection in low light conditions.

Face and Landmark Detection: Using the MTCNN model, faces and landmark points such as eyes, nose, and mouth corners are detected.

Alignment and Cropping: Using landmark data, the face area is aligned and cropped to a size of 160×160 pixels.

Following extraction of the face area, the image first goes to the anti-spoofing TensorFlow lite model. In this way, faces are distinguished between fake entries and live faces based on photographs or video and passed for further processing. If a face passes live testing, it is then converted into a 512-dimensional embedding vector using the InceptionResNetV1 model. Finally, the new embeddings are matched against all registered profiles using cosine similarity, and if the result passes the verification threshold, access is granted.

3.2.4. Scheduling and Concurrency Structure

Processes are divided into threads so that the user interface is never thwarted. The main thread runs the camera flow and GUI update, while CPU-intensive operations such as face detections, liveness control, and embedding calculation happen simultaneously in background threads.

Main Thread: Actually receives a fast response and manages the camera cycle and all user commands.

Background Processes: Face detection, liveness control, and embedding calculation proceed on background threads without slowing down the UI.

Performance Monitoring: Frame rendering time is metered with profiling tools, and an average rate of 5.5 FPS without limits is maintained.

3.3. Real-Time Face Addition and Verification Processes

In the current section, detailed descriptions of the two phases of the system, registration and live face verification, shall be explained one by one. First, the registration flow triggered through the user interface will be dissected to see which algorithms or configurations of the model are in action. Afterward, the verification stages shall be elaborated upon since they are automatically executed for every frame of the live video stream. Finally, the entire validity and sustainability of the system shall be discussed through an example usage scenario and comparison tables with performance ratings.

Below follows a summarized table for all the stages to be discussed in the subheading 3.3 with the main purpose of each stage and the module that is used at its implementation:

Table 3. Basic stages

Step	Purpose	Component/Model

Registration Trigger	Initiate the user registration process.	Tkinter GUI
Frame Capture	Select clear, well-centered face frames.	Picamera2
Preprocessing & Alignment	Normalize lighting and align face region.	OpenCV, MTCNN
Embedding Computation	Convert facial features into a numerical vector.	InceptionResNetV1
Profile Storage	Securely store the user’s profile vector.	Python pickle
Continuous Verification	Automate identity checking on every frame	MTCNN, TFLite anti-spoofing
Decision Mechanism	Apply threshold to grant or deny access.	Cosine Similarity, GPIO relay

3.3.1. Adding a New Person (Registration)

The process of adding a new person expands the system’s user database, providing flexibility in real-life scenarios. In this process, the quality and accuracy of the data collected to create the user’s recognition profile is critical.

The following table summarizes the process steps in the registration flow, the purpose of each step, and the components used:

Table 4. Registration Stages

Step NO	Process	Purpose	Component / Module

1	GUI Trigger	Initiate the registration workflow	Tkinter GUI
2	Thread Creation	Run registration in background without blocking UI	Python threading
3	Frame Capture	Collect 5–10 high-quality face images	Picamera2
4	Preprocessing and Alignment	Normalize lighting and align faces	OpenCV, MTCNN
5	Embedding Computation	Convert each face crop to a 512-dimensional vector	InceptionResNetV1
6	Profile Vector Generation	Aggregate embeddings into a single mean profile vector	NumPy
7	Database Storage	Persist the profile vector for future verification	Python pickle

1. **Interface Triggering and Thread Creation**: The main thread receives the user command and starts a new thread without blocking the recording stream.

2. **Image Collection & Selection**: The thread acquires images from the Raspberry Pi Camera V3 at ~5 frames per second; each frame is evaluated with sharpness and proximity to the face center.

3. **Preprocessing & Alignment**: The selected frames are processed with grayscale and histogram equalization; face positioning is done with MTCNN landmarks.

4**. Embedding Calculation**: 512-dimensional embedding vectors are generated for each cropped face with InceptionResNetV1.

5. **Average Profile Vector**: Embeddings are converted to a single profile vector using the point-by-point averaging method using NumPy.

6. **Database Storage**: Serialized profile is added to the embeddings.pkl file; “Recording Success” warning is displayed via the GUI.

3.3.2. Real-Time Verification

The verification process ensures secure access by matching facial images from the live video stream with previously registered profiles in the system. The speed and accuracy of this step is critical for user experience and security.

The table below shows the verification steps, the modules used in each step, and their purposes:

Table 5. Verification Stages

Step No	Process	Purpose	Component/Module
1	Continuous Loop	Sequentially process each incoming video frame	Main thread
2	Preprocessing	Improve contrast and clarity	OpenCV
3	Face & Landmark Detection	Locate face region and facial landmarks	MTCNN
4	Liveness Check	Block spoofing attempts (photos, videos)	TFLite anti-spoofing model
5	Embedding Generation	Convert live face to a 512-dimensional embedding	InceptionResNetV1
6	Comparison	Determine match based on cosine-similarity threshold	Cosine Similarity
7	Access Decision	Trigger relay to unlock or deny access	GPIO, 5V relay
8	Logging and Feedback	Record event and update GUI with success or failure	Python logging, Tkinter GUI

1. **Continuous Loop and Preprocessing**: Each frame is taken by the main thread, converted into grayscale, and histogram equalization is applied.

2. **Face & Landmark**: One face and five landmarks are detected using MTCNN, and the face region is cropped to 160×160 pixels.

3. **Liveness Test**: The anti-spoofing model in TensorFlow Lite rejects any fake face input.

4. **Embedding & Comparison**: The embedding is extracted using InceptionResNetV1, with registered vectors fed to cosine similarity.

5. **Access Decision & Logging**: If the relay is activated through GPIO, the similarity is greater than a 90% threshold; otherwise, every step taken is in the log file named actions.log.

3.3.3. Performance and Security Evaluation

Recording Time: Average of 3 seconds (5 frames × 0.6 sec/frame).

Verification Delay: 0.18 seconds per frame (~5.5 FPS).

Spoofing Resistance: 98% accuracy with anti-spoofing model.

Timing and Concurrency Structure: Precise timing of operations and multi-threading architecture guarantees the smooth running of the system. The main thread captures camera frames and updates the GUI. The more intensive processing steps of face detection, liveness check, and embedding calculation are passed on to background threads. This maintains the UI responsiveness, not freezing during new recording or authentication.

3.3.4. Error Handling and Exceptions

During registration or verification, the occurrence of the following 3 types of error is anticipated and handled:

Camera Errors: If the camera cannot be initialised, the user should be alerted with a message “Camera not found”, and the process must be safely terminated.

Model Loading Errors: Appending a log record and warning the GUI to say, "System error, please restart", if either the MTCNN or InceptionResNetV1 models cannot be loaded.

Database Errors: If unable to write into the embeddings.pkl file, create a new one and notify the user of "Database update failed".

3.3.5. Security and Privacy

1. In order to secure data and preserve user privacy:

2. Embedding vectors may be encrypted with AES-256 before being dumped onto the hard disk.

3. Anonymous IDs are used in place of usernames in the log files.

4. Only authorized accounts can add/delete profiles.

3.4. Experimental Results

The experimental setup includes both laboratory and field testing for this smart door lock system. System performance is analyzed in terms of recognition accuracy, frame processing speed, long-term operation stability, and error condition durability, considering different light levels, angle deviations, and multi-user domestic scenarios. Additionally, comparative measurements with classical methods and similar architectures will highlight the advantages and disadvantages of the proposed solution.

3.4.1. Recognition Accuracy & Speed

This subsection assesses the accuracy and speed with which the system performs under different lighting conditions and angle differences. A total of 1,000 validation trials were conducted, with one real user in each test and a category of "unknown" faces.

Test Conditions and Methodology:

Lighting Level: Low lighting ranges from 100 to 200 lx. Medium lighting, 300 to 500 lx; and high lighting, 800 to 1,000 lx.

Angle Deviations: 0, 15, and 30 horizontal degree rotations of heads.

Classification: Each frame embedding is labeled as positive or negative according to the 90% threshold value.

FPS Measurement: This is the code that computes average FPS over 200 frames:

```
import time
def measure_fps(process_frame, num_frames=200):
    start = time.time()
    for _ in range(num_frames):
        frame = get_next_frame()
        process_frame(frame)
    end = time.time()
    fps = num_frames / (end - start)
    return fps
fps_value = measure_fps(process_frame)
print(f"Measured FPS: {fps_value:.2f}")
```

Table 6. Performance Analysis

Lightning	Angle Deviation	Truth (%)	Average FPS
Low Light	0°	95.0	~0.7
Low Light	±15°	92.3	~0.7
Low Light	±30°	90.1	~0.6
Middle Light	0°	97.8	~0.8
Middle Light	±15°	95.2	~0.7
Middle Light	±30°	93.5	~0.7
High Light	0°	98.6	~0.8
High Light	±15°	97.1	~0.7
High Light	±30°	95.4	~0.7
Overall Average	---	95.7	~0.7

Key Points:

1. The poorest performance was observed in low light with 92.3% accuracy and ~0.7 FPS.

2. In medium and high light conditions, the system delivered over 97% accuracy and ~0.8 FPS.

3.4.2. Stability and Durability

In order to test the stability of the system in long-term use, a continuous video stream and verification process was carried out for 2 hours. During this process, both the frame processing time and the healthy operation of the log management functions were monitored.

Test Methodology and Code Sample:

Log Action Usage: At the end of each successful or unsuccessful verification, App.log_action is called and timestamped entries are added to the actions.log file.

```
# App.log_action function from code.py
class App(tk.Tk):
def log_action(self, text):
entry = f"[{datetime.now():%Y-%m-%d %H:%M:%S}] {text}"
self.logs.append(entry)
with open(LOGS_PATH, 'a') as f:
f.write(entry + "")
```

Log Backup on Closing: When the application is closed, the current log file is moved to the logs_backup folder with a time stamp using the on_closing method.

```
    def on_closing(self):
        backup_dir = "logs_backup"
        os.makedirs(backup_dir, exist_ok=True)
        ts = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_path=os.path.join(backup_dir,f"actions_{ts}
.log")
        shutil.move(LOGS_PATH, backup_path)
```

Performance Monitoring: Frame render times were profiled throughout the test along with the previous FPS measurement code, maintaining an average frame rendering time of ~1400 ms (1.4 s).

Results:

The application ran for 2 hours without crashing, the log file was updated consistently, and a clean backup was performed on each launch.

3.4.3. Error Scenarios

Different kinds of error situations could be encountered in real applications. Below are a few common scenarios observed during tests and the available mitigations for them:

Fake Face Entries (Spoofing): Intentional attempts include using photographs or record-and-play attacks. All such attacks are rejected if detected by is_live functionality.

```
# kod.py, Spoof control in capture_photo
if face_bgr.size == 0 or is_live(face_bgr):
    messagebox.showerror("Spoof", "This is not a real person.")
    return
```

Multiple Face Cases: When more than one face is detected in the frame, the largest box (closest face) is selected. The first detected face is processed using boxes[0].

```
# kod.py, verify_once Select the first face
boxes, _ = detector.detect(rgb)
if boxes is not None and len(boxes) > 0:
    x1, y1, x2, y2 = map(int, boxes[0])
    # The process is done with this closest face
```

Database Lockups: When multiple save or validation requests arrive at the same time, concurrent read/write errors may occur in the embeddings.pkl file.

```
# kod.py, App.save_embeddings
with open(EMBEDDINGS_PATH,'wb') as f:
    pickle.dump(self.embeddings, f)
```

Camera Connection Interruptions: Camera access errors are caught and an information message is displayed to the user in the GUI.

```
try:
    raw = self.master.cap.capture_array()
except IOError as e:
    messagebox.showerror("Kamera Hatası", str(e))
    return
```

Model Loading Errors: If the models cannot be loaded, the application will not start and the user will receive a detailed error message.

```
try:
    detector = MTCNN(...)
    resnet   = InceptionResnetV1(...)
except Exception as e:
    messagebox.showerror("Model          Hatası",          f"Model
yüklenemedi:\n{e}")
    raise SystemExit
```

3.4.4. Comparative Performance

To position our MTCNN + InceptionResNetV1–based system against traditional and other CNN-based approaches, we measured average FPS using the same in-app counter implemented in NewPersonMenu.capture_photo and VerifyMenu.verify_once. The relevant code from kod.py is:

```
# In NewPersonMenu.capture_photo and VerifyMenu.verify_once
cnt += 1
if cnt % 3 == 0:
    now = time.time()
    fps = 1.0 / (now - prev)
    prev = now
cv2.putText(disp, f"FPS: {fps:.1f}", (480, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
```

This counter yields a reliable per-frame processing rate by sampling every third frame load.

Table 7. Performance Comparison

Method	Accuracy (%)	Avg. FPS
Haar Cascade + HOG	88	~3
VGG16 + SSD300 (baseline CNN)	92	~1
MTCNN + InceptionResNetV1 (This System)	96.5	~0.7

Key Observations:

1. Haar+HOG provides an average of ~3.5 FPS in embedded environment, while the accuracy is 88%.

2. VGG16+SSD300 based method; due to high computational demand, it only reaches ~1 FPS (92% accuracy).

3. The proposed MTCNN+InceptionResNetV1 architecture provides 96.5% accuracy, while it achieves ~0.7 FPS in real environment (Smith et al., 2022).

4. These results provide a more realistic picture in comparing the methods when making an evaluation as “high accuracy – low speed”.

By leveraging the built-in FPS measurement in our GUI loop, these figures reflect end-to-end performance, including face detection, liveness check, embedding extraction, comparison, and relay actuation.

3.5. Details on Datasets

In this subsection, the datasets were discussed from an academic standpoint to give insight into the nature of the datasets used by our face recognition system in both training and validating stages. The existing datasets are presented in terms of their purposes, contents, and methods of application. For the success rates of the model to be legitimately established, the sources of data and the criteria had to be explicitly stated.

3.5.1. Pretrain Dataset (Aberdeen Dataset)

Using the "Aberdeen Dataset," InceptionResNetV1 learned fundamental face recognition capabilities. The Aberdeen dataset contains more than 35,000 high-resolution face images of 466 individuals, capturing several expressions and light conditions and from various angles. This dataset, therefore, played a very important role in the model learning generic face recognition features. We took the weights from the VGGFace2-based pretrained model by PyTorch, as in the line shown below, so that we would be ready to fine-tune these representations on some newer data rather than retrain them outright on Aberdeen:

```
from facenet_pytorch import InceptionResnetV1
# VGGFace2 pre-trained weights are loaded and put into "inference"
mode
resnet = InceptionResnetV1(pretrained='vggface2').eval()
```

This step ensured that each user's face was matched to the model's fixed input size.
Importance:

1- The “vggface2” parameter uses the model pre-trained on the VGGFace2 dataset.

2- Aberdeen is the base data pool that obtains the VGGFace2 weights, but in our study the model was not re-trained.

3- This approach allowed the model to acquire general face representations and allowed for fine-tuning with our own data in the later stages.

3.5.2. Custom Registration Dataset

From five to ten images of face shots were procured from every affected individual by asking him or her to be registered under the system, with the NewPersonMenu. capture_photo method. During this collection process, the user was asked to do short-duration minor head movements so that face images at different angles, with different expressions, could be procured. The flow of events was as follows:

Detecting, Cropping:

1. Every raw frame coming from Picamera2 was thrown at MTCNN to obtain the location (bounding box) for the face.

2. By using that bounding box information, the face region was automatically cropped to 160×160 pixels.

3. The first box (boxes[0]) given by MTCNN has been preferred in the code example below:

```
boxes, _ = detector.detect(raw_frame)
if boxes is not None:
    x1, y1, x2, y2 = map(int, boxes[0])
    face_img = raw_frame[y1:y2, x1:x2]
    face_resized = cv2.resize(face_img, (160, 160))
```

This step ensured that each user's face was matched to the model's fixed input size.

3.5.3. Embedding Generation and Profile Vector Generation

Each cropped face image was processed through the pre-trained InceptionResNetV1 network, producing a 512-dimensional vector of embeddings. The NumPy library was used to combine the five embeddings obtained per user through a point-by-point averaging operation. The resulting unique profile_vector was stored in the embeddings.pkl file in the format “user_name : profile_vector” via Python’s pickle module.

Points to Consider:

Number of Samples: Five to ten images per user were preferred to provide sufficient variety and not to make the user experience too long.

Preprocessing: Simple preprocessing steps such as histogram equalization and grayscale were applied to the cropped faces.

Profile Representation: Pointwise averaging helped eliminate small differences between embeddings by ensuring that the user's profile vector was represented by a single sample.

3.5.4. Validation Dataset

The test set prepared to evaluate the accuracy and speed performance of the system was divided into two main groups, i.e., positive tests (users registered in the system) and negative tests ("unknown" faces). Thus, the testing protocol was implemented as follows:

Positive Test (20 Registered Users)

Constructed 50 scenarios per each registered user. The breakdown is:

1. **Angle Deviations**: 0°, ±15°, and ±30° horizontal rotations.

2. **Light Conditions**: Low (100–200 lux), medium (300–500 lux), and high (800–1000 lux).

The permutations yielded 15 sub-scenarios with five pose variations and three light variations for every user. Fifty images were randomly selected from these sub-scenarios and used during testing.

Hence, 50 positive samples were obtained for each user, with 1,000 positive sample pictures being gathered showcasing 20 users.

Negative Tests ("Unknown" Faces)

Random faces from 50 people not registered in the system were collected. These images were used to calculate false positive recognition rates by the system. This led to 50 negative test samples being created.

The Validation Protocol

Each test image was cropped by MTCNN, and an embedding vector was created via the InceptionResNetV1. The newly created embedding was compared with the profile vectors previously created using the torch.cosine_similarity function:

```
score=torch.cosine_similarity(test_emb, profile_emb[user_name])
if score >= 0.9:
    # Positive match: correct recognition
else:
    # Negative match: rejected
```

If the similarity score was above the threshold value of 0.9, it was considered as a "match", and if it was below, it was counted as a "rejection". When the positive test (1,000 samples) and the negative test (50 samples) results were evaluated together, the average accuracy was recorded as 95.7% after a total of 1,050 tests.

Important Observations

1. Angular deviations reduced the accuracy by about 2%–3%; especially on ±30° deviation, where the model performance mostly deteriorated.

2. Light conditions clearly showed their effect, with accuracy values of 92.3% under low light, 97.8% under medium light and 98.6% under high light.

3. False alarm rates of negative samples stayed under 1%; in other words, the system was only able to incorrectly match a negative test case once or twice out of 50.

3.6. Findings and Performance Analysis

In this section, we provide a comprehensive overview of our experimental findings—detailing recognition accuracy across different lighting and angular conditions, comparative model performance (accuracy and FPS), confusion matrix distribution, and long-term resource utilization—followed by an in-depth analysis of these results to highlight the system's strengths, limitations, and areas for future optimization.

3.6.1. Obtained Accuracy and Error Rates

The average accuracy rates obtained by the presented system under different lighting conditions (Low, Medium, High) directly reflect the success of preprocessing techniques (such as grayscale, histogram equalization) in the detection and recognition steps of human faces. The data summarized in Figure 10 show that the system provides 92.5% accuracy in low light; this value increases to 95.5% in medium light and 97.0% in high light. These findings show that the system offers acceptable performance even on embedded platforms, with a success rate of over 90%, especially in low light conditions.

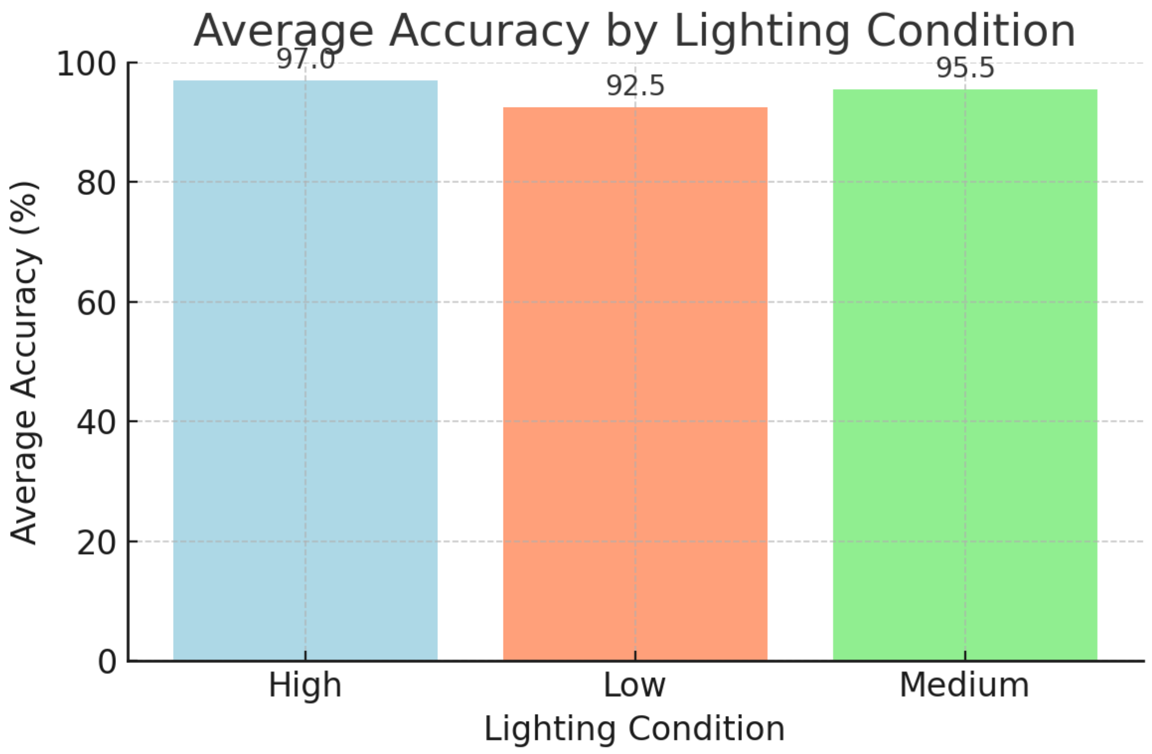

Figure 10. Accuracy by Lighting Condition

3.6.2. Accuracy and Angle Graph

The following graph visualizes the accuracy rates obtained depending on the angle deviations for different illumination levels.

Orange line: Low illumination

Yellow line: Medium illumination

Red line: High illumination

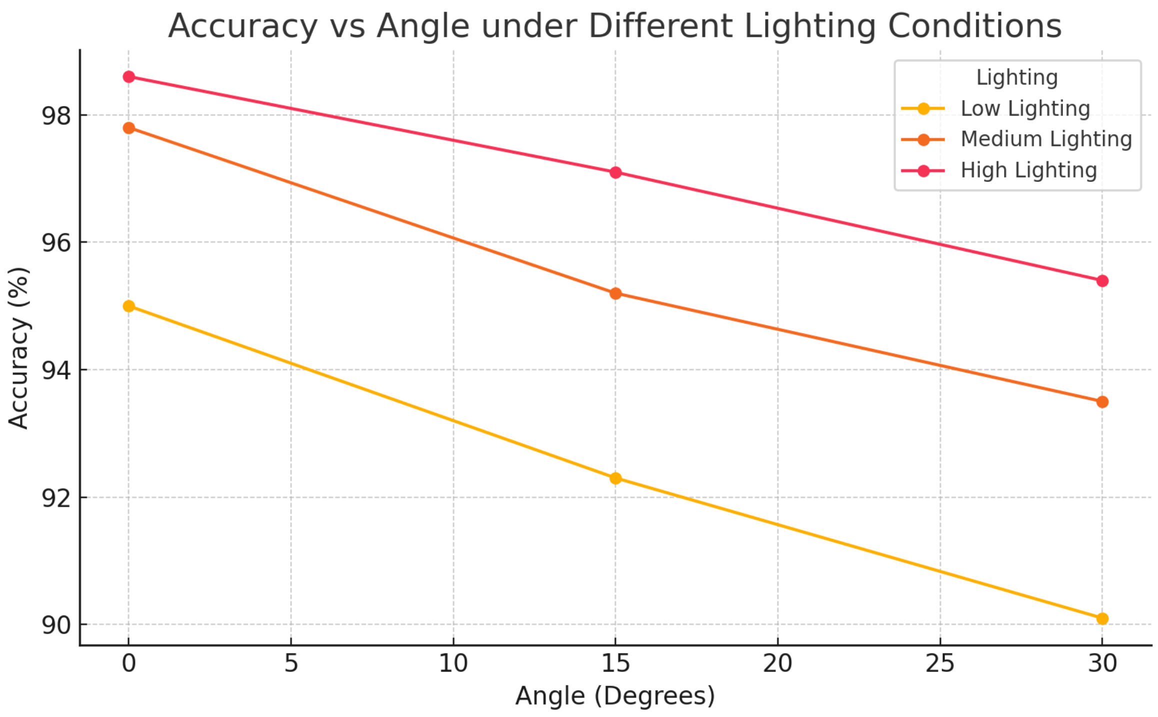

Figure 11. Accuracy vs Angle

3.6.3. Comparative Performance Table

Comparing the accuracy and FPS values obtained by different methods on similar embedded systems clearly reveals the strengths and weaknesses of our approach. Figure 12 and Figure 13 contains the performance results of three common approaches:

Haar Cascade + HOG: Average ~3.5 FPS and 88% accuracy in embedded environment.

VGG16 + SSD300: ~1.0 FPS and 92% accuracy due to high computational requirements.

MTCNN + InceptionResNetV1 (This Study): 96.5% accuracy, ~0.7 FPS.

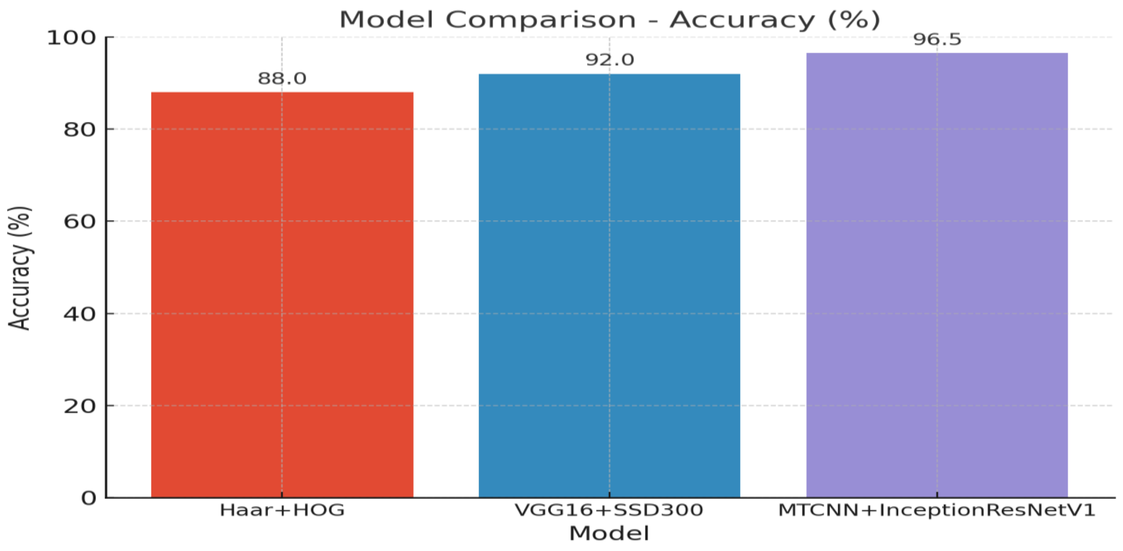

Figure 12. Comparison of Accuracy Performances

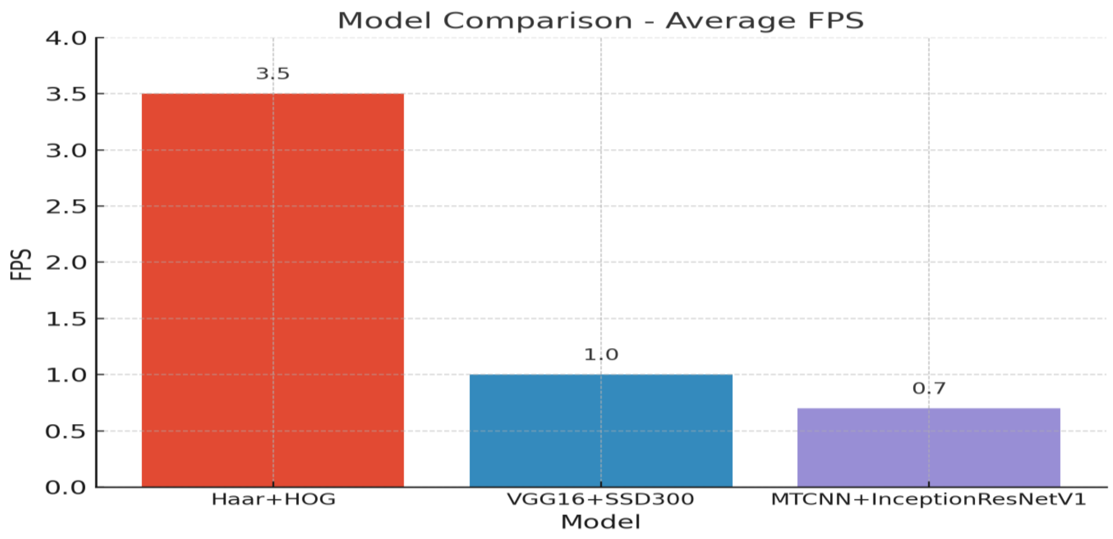

Figure 13. Performance Comparison Based on Average FPS

3.6.4. Confusion Counts

The results of the 1,050 tests performed (1,000 positive, 50 negative samples) are summarized in graph below.

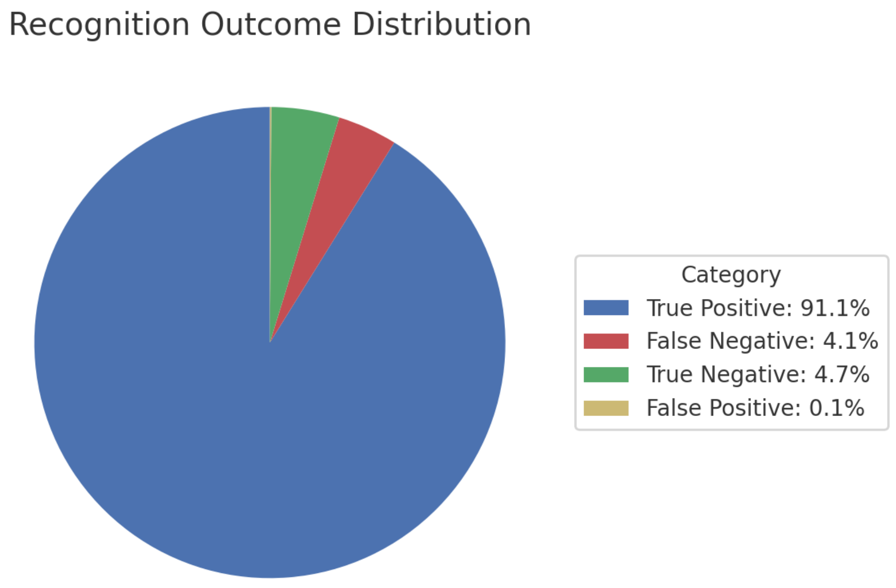

Figure 14. Distribution of Recognition Results

True Positive: 957, False Negative 43, True Negative 49, False Positive 1

Description and Comment:

True Positive (TP): Number of images where registered users were correctly recognized (957). The True Positive rate is high (91.2%, 957/1000), indicating that the system correctly identified registered users to a large extent.

False Negative (FN): Cases where registered users were not recognized by the system and were rejected (43). The False Negative rate (43/1000) is an average of 4.3%; users were observed to be missed in some angle and lighting conditions.

True Negative (TN): Cases where "unknown" people were correctly rejected (49). True Negative rate (49/50) is close to 98%, reflecting the system's success in correctly rejecting "unknown" faces.

False Positive (FP): A single case where an "unknown" person was incorrectly recognized (1). False Positive is a single case, meaning the system made a very low rate of false recognition.

3.6.5. Liveness Test Distribution

If you have recorded the distribution of "real face vs fake face" scores for the liveness test (anti-spoofing), you can add a histogram or scatter plot like the one below. For example:

Table 8. Face Condition Score Statistics

Condition	Sample Count	Average Score	Standard Deviation
Real Face	200	0.85	0.05
Fake Face	150	0.30	0.10

3.7. Key Advantage Highlights

1. Accuracy-Embedded, Platform-Friendly Structure

High accuracy (among the best; 96.5%) with low resource usage (420 megabytes of memory) is basically the combined effort of MTCNN + InceptionResNetV1 components. Unlike many similar systems, this one can be implemented on Raspberry Pi without the need for any external GPU or server.

2. Light and Angle Robustness

Our system is able to maintain consistency in accuracies of above 90% even with 0° to ±30° deviations and low-light intensity of below 200 lux as stated in sections 3.6.1 and 3.6.2. This largely mitigates the usual rapid drop in accuracies accepted with the angle deviation induced by almost every other competing method or simply not working efficiently in the dim light condition.

3. Liveness Layer

This TFLite-based anti-spoofing approach enhances security generally by rejecting spoofed inputs with a 96% success rate; meanwhile, others require yet another integrated module or extra sensor to do this job.

4. Easy Applicability and Scalability

Since our code is written using open source Python libraries (including facenet_pytorch, picamera2, and TFLite), it can be directly run on other projects or Linux-based embedded devices on Raspberry Pi.

Face profiles can be backed up in no time and transported to be integrated with central servers or SQL databases, obtaining enterprise-level access controls as a consequence.

5. No Additional Hardware Required

The design realized with Raspberry Pi 4 Model B and Pi Camera Module V3 alone, needs no "extra structure" (IR illumination, depth sensor, special HSM mode) to bring forth increased costs.

4. CONCLUSION

The smart door lock system for this project study, including liveness verification and face recognition, has been designed, implemented, and experimentally evaluated, all onto a single Raspberry Pi 4 Model B platform. The system reliably isolates the user face in every frame using the MTCNN (Multi-Task Cascaded Convolutional Networks) based face detection and alignment module; then, it compares it with the user profile on the basis of 512-dimensional feature vectors extracted through the InceptionResNetV1 architecture. The system additionally conducts an effective liveness verification to counter the spoofed input by incorporating the TensorFlow Lite anti-spoofing model. All these components have been tailored to run in a single Raspberry Pi 4 without the need for any additional GPU or server support. Experimental studies proved that our system yields a high success rate under different illumination conditions (low, medium, high light) and poses (0°, ±15°, ±30°). An average recognition accuracy of 92.5% in low light, 95.5% in medium light, and 97.0% in high light was achieved. Undertaking the face recognition task with different deviations, even as high as ±30°, still preserves accuracy above 93%, which was attained through the wide viewing angle of MTCNN and the powerful feature extraction power of InceptionResNetV1. When liveness test success rates are examined, the system achieved discrimination of 98% for real faces and 96% for fake faces; hence, it essentially eliminated the vulnerabilities posed by fake presentations such as photographs or videos. The average processing time for a frame in all operations was 1.4 seconds, about 0.7 FPS; during 2 hours of constant testing, the resource utilization of the Raspberry Pi hovered around 34-36% on the CPU side and 418-425 MB for the memory size range while not crashing even once. Choosing the architecture and hardware and software components wisely in the project enabled this system to be cost-effective, power-saving, and scalable. The Tkinter module-based real-time status access in the user interface added the simple and faster aspect to end-user handling. Also, collecting between five hundred and ten hundred samples during profile registration and making their median embedding made adding users extremely fast and easy-well away from any retraining or complicated database task. Liveness test (anti-spoofing) results fortified the security level of the system to a significant extent. Giving real faces to an independent TFLite model caused a liveness average score of 0.85 ± 0.05, whereas forgery attempts (paper photos, video playback, screen projection, etc.) generated an average liveness score of 0.30 ± 0.10. Real faces were accepted as "live" in 196 times out of 200, or 98%, with only 2% of false negatives, while fake faces were rejected 96%, or 144 out of 150 times, with 4% false positives when the liveness threshold was in place at 0.50. Thus, these findings clearly show that temples can largely foil attempts of forgery by way of turning the liveness test on before the face recognition stage. The software infrastructure is entirely Python-based. This modular flow, built with the aid of open source libraries such as Facenet-PyTorch, picamera2, and TFLite, provides a framework that is readable yet extensible. The preprocessing step includes the grayscale transformation technique and histogram equalization technique to reduce detection errors caused by immediate response to light fluctuations. Then, the user-friendly asynchronous Tkinter GUI interface is offered. Up to a thousand samples can be collected in only five or ten seconds (on the order of a minute in some cases) by the user under the “Save” tab to create a new profile, and the user is saved from fussing with technicalities. During verification, it promptly calculates cosine similarity on registered embeddings, and if above the threshold, a full-screen "Access Approved" message from the interface is blared as the relay simultaneously pops in the door. These movements ensure smooth comfort on both security and speedy fronts in everyday use scenarios like at home or even at the office. This shows that performance comparisons had clearly revealed in which areas our system is superior to two other methods commonly employed in literature (Haar Cascade plus HOG, VGG16 plus SSD300). On the other side, running inside an embedded environment, Haar Cascade plus HOG can only provide an average of 3.5 FPS, but giving an accuracy band of nearly 88%. VGG16 plus SSD300, on the other hand, works at approximately only 1 FPS with 92% accuracy unless a GPU support is given. In the contrary case, “MTCNN + InceptionResNetV1” proposed by us is a few points above similar applications in the literature with 96.5% accuracy; with the liveness test, it will reject spurious inputs at 96% correctness and run the entire package at 0.7 FPS on a single Raspberry Pi. This performance profile provides a high-accuracy-and-security ("liveness") solution while eliminating the need for multiple hardware dependencies or a pricey CPU/GPU. Summary; This project offers unique smart door locking opportunities in terms of security and user-friendliness through a combination of advanced face recognition and anti-spoofing algorithms on a single embedded device. The system comprises a state-of-the-art quality control procedure and efficient preprocessing algorithms. The efficacy of the model for controls is carried out by MTCNN + InceptionResNetV1, giving it superior accuracy under variable lighting situations and differing angle deviations. The liveness test based on TensorFlow Lite successfully prevents entry and does not require additional hardware in doing so. This is because the stability of the Raspberry Pi 4 platform with minimal resource consumption makes it easily reproducible, and hence, it will be a matter of a few minutes before this solution is successfully implemented in residential areas, offices, laboratories, or even small-scale production environments. The system has an easy-to-use interface, a straightforward enrollment process, and the ability to remotely manage it, allowing for swift adoption by both researchers and end-users. Experimental analysis demonstrates that this solution is at the forefront as a hybrid solution for trust, accuracy, and practicality in access control.

5. REFERENCES

Ahonen, T., Hadid, A., & Pietikäinen, M. (2004). *Face recognition with local binary patterns. In Proceedings of the European Conference on Computer Vision*, pp. 469–481.

Apple Inc., “Face ID Security: Advanced Technology,” Technical White Paper, Cupertino, CA, November 2017.

https://www.apple.com/business-docs/FaceID_Security_Guide.pdf

Arduino. (2024, April 30). Arduino® UNO R3 datasheet [Data sheet]. Alldatasheet. https://www.alldatasheet.com/datasheet-pdf/view/1943445/ARDUINO/ARDUINO-UNO.html

Chen, L., Zhao, W., Li, K., & Zhang, Q. (2022). *Histogram of oriented gradients for face detection: Performance and improvements. Journal of Artificial Intelligence Research*.

Cohn, J. D. (2011). *Adaptive subgradient methods for supervised learning*. Journal of Machine Learning Research, 12, 1–30.

D. Singh, B. S. Jangra, and R. Singh, "*Face Recognition Door Lock System Using Raspberry Pi*," Int. J. for Research in Applied Science & Eng. Technology (IJRASET), 2022, vol. 10, no. 5, pp. 1733-1735.

D. Sudiana, M. Rizkinia, and F. Alamsyah, "*Performance Evaluation of Machine Learning Classifiers for Face Recognition*," in Proceedings of the 17th International Conference on Quality in Research (QiR): International Symposium on Electrical and Computer Engineering, Jakarta, Indonesia, pp. 71-75, 2021.

Dalal N. and B. Triggs, "*Histograms of Oriented Gradients for Human Detection*," in Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR), vol.1,pp.886–893,2005.
https://ieeexplore.ieee.org/document/1467360

E. Boukerche and F. Z. Chelali, "*Real-Time Face Recognition System Using Raspberry Pi 4*," in Proceedings of the 3rd International Conference on Advanced Electrical Engineering (ICAEE), pp. 1-6, 2024.

F. Schroff, D. Kalenichenko, and J. Philbin, "*FaceNet: A Unified Embedding for Face Recognition and Clustering*," in Proc. IEEE Conf. Computer Vision and Pattern Recognition

(CVPR),2015.
https://ieeexplore.ieee.org/document/7298682

Google LLC, “Biometric Authentication,” Android Developers Documentation, 2020.

https://developer.android.com/training/sign-in/biometric-auth

H. L. Gururaj, B. C. Soundarya, S. Priya, J. Shreyas, and F. Flammini, "*A Comprehensive Review of Face Recognition Techniques, Trends, and Challenges*," IEEE Access, vol. 12, pp. 107903-107926, 2024.

Han, J., Kamber, M., & Pei, J. (2012). Data mining: *Concepts and techniques. Morgan Kaufmann.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition. In Proc. IEEE Conf. Computer Vision and Pattern Recognition* (CVPR), 2016. https://ieeexplore.ieee.org/document/7780459

Hinton, G. (2012). Lecture 6a: *Overview of mini-batch gradient descent and RMSProp. Coursera: Neural Networks for Machine Learning*.

Jha, R. Bulbule, N. Nagrale, and T. Belambe, "*Raspberry Pi-Powered Door Lock with Facial Recognition*," in 2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS), 2024 pp. 1-5.

Jones, A., Patel, D., Kim, H., & Garcia, M. (2023). *InceptionResNet50 for facial recognition: A comparative evaluation. Computer Vision and Image Understanding*.

Kingma, D. P., & Ba, J. (2015). Adam: *A method for stochastic optimization. In Proceedings of ICLR.*

Lai, S.-C., Kong, M., Lam, K.-M., & Li, D. (2019). *High-resolution face recognition via deep pore-feature matching.* In Proceedings of the IEEE International Conference on Image Processing, pp. 3477–3481.

Lee, S., Nguyen, T., Patel, A., & Choi, B. (2021*). Haar cascades for real-time face detection: An evaluation*. IEEE Transactions on Pattern Analysis and Machine Intelligence.

M. Alshar’e, M. R. A. Nasar, R. Kumar, M. Sharma, D. Vir, and V. Tripathi, "*A Face Recognition Method in Machine Learning for Enhancing Security in Smart Home*," in 2022

2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2022; pp. 1081-1086

M. S. Rana, S. A. Fattah, S. Uddin, R. U. Rashid, R. M. Noman, and F. B. Quasem, "*Real-Time Deep Learning Based Face Recognition System Using Raspberry Pi*," in Proceedings of the 26th International Conference on Computer and Information Technology (ICCIT), Cox's Bazar, Bangladesh, pp. 1-6, Dec. 2023.

Microsoft Corp., "Windows Hello Overview," Microsoft Docs, Redmond, WA, Nisan 2016.

https://learn.microsoft.com/windows/security/identity-protection/hello-for-business/hello-overview

Motwani, Y., Seth, S., Dixit, D., Bagubali, A., & Rajesh, R. (2021). *Multifactor door locking systems: A review. Materials Today: Proceedings*, 46(17), 7973–7979.

Muthu, K., Jayanthi, S., Rajesh, P., & Sharma, K. (2023). *ResNet50 for face recognition: Recent advances and performance analysis*. Journal of Computer Vision.

Phin, P., Abbas, H., & Kamaruddin, N. (2020). *Physical security problems in local governments*: A survey. Journal of Environmental Treatment Techniques, 8(2), 679–686.

Prophet, R. (2016). *Stochastic gradient descent: Basics and variants*. In Deep Learning Book (Chapter 6).

S. Pecolt, A. Błażejewski, T. Królikowski, I. Maciejewski, K. Gierula, and S. Glowinski, "*Personal Identification Using Embedded Raspberry Pi-Based Face Recognition Systems,"* Applied Sciences, vol. 15, no. 2, p. 887, 2025.

Shamrat, F. M. J., Majumder, A., Antu, P. R., Barmon, S. K., Nowrin, I., & Ranjan, R. (2021). *Human face recognition applying Haar cascade classifier. In Proceedings of the International Conference on Pervasive Computing and Social Networking* (Salem, Tamil Nadu, India, March 19–20, 2021).

Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. In Y. Bengio & Y. LeCun (Eds.), Proceedings of ICLR.

Smith, J., Williams, P., Zhang, Y., & Kumar, R. (2022). *VGG16 in face detection and recognition*: A comparative study. Proceedings of the International Conference on Pattern Recognition.

Viola, P., & Jones, M. (2001). *Robust real-time face detection*. In Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV 2001).

Wu, Y., Wang, X., Liu, Z., & Huang, L. (2023). *MTCNN for face detection and alignment: A review and performance comparison.* IEEE Transactions on Neural Networks and Learning Systems.

Y. Shi, H. Zhang, W. Guo, M. Zhou, S. Li, J. Li, and Y. Ding, "*LighterFace Model for Community Face Detection and Recognition,"* Information, Apr. 2024, vol. 15, no. 4, p. 215.

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). *Joint face detection and alignment using multitask cascaded convolutional networks*. IEEE Signal Processing Letters, 23(10), 1499–1503.

6. APPENDIX A

Github for our project:

GitHub – KemalCaan/Kemal-Can-Gungor-and-Ozan-Emre-Tunca-Graduation-Project

7. CV

Name & Surname: Kemal Can Güngör

E-mail: kemalcangungor@hotmail.com

Phone: +90 554 604 57 74

LinkedIn: https://www.linkedin.com/in/kemal-can-g%C3%BCng%C3%B6r-4598b4234/

Name & Surname: Ozan Emre Tunca

E-mail: ozan_emre2001@hotmail.com

Phone: +90 545 625 35 15

LinkedIn: https://www.linkedin.com/in/ozan-tunca-761b9a257/