UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Face Recognition based Door Lock

Author:
Maksym BILYK

Supervisor:
Dmytro PRYIMAK

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

I, Maksym BILYK, declare that this thesis titled, "Face Recognition based Door Lock" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Face Recognition based Door Lock

by Maksym BILYK

Abstract

Nowadays most modern systems/services are distributed and scalable to be able to meet the growing amount of data and users per product. Therefore, this thesis is related to the development of a multi-user distributed smart Face Recognition-based home lock.

This system should provide the possibility for users to trigger the Face Recognition-based authentication process without direct contact with the lock, which should be comfortable for the user. Also, the system should support the enrollment of multiple users.

Acknowledgements

I want to thank my supervisor Dmytro Pryimak for his pieces of advice and genuine interest in the result of this project. I am deeply grateful to my friends and family for their support and help in every way possible.

Contents

List of Figures

List of Abbreviations

FR	**F**ace **R**ecognition
HGR	**H**and **G**esture **R**ecognition

Dedicated to my parents

Chapter 1

Introduction

1.1 Biometrics-based security systems

1.1.1 Overview

Biometric-based security systems have gained significant importance in recent years due to their ability to provide more reliable and efficient authentication and access control methods. Traditional security measures like passwords, PINs, or physical access cards have limitations, such as being quickly forgotten, shared, or stolen. Biometric-based systems leverage unique physiological or behavioral characteristics, such as fingerprints, facial recognition, iris scanning, or voice patterns, for identification purposes.

1.1.2 Post COVID-19

Since November 2019, COVID-19 has forced many people to change their behavior to avoid spreading the virus. One of the most important measures is social distancing, which involves keeping a safe distance from others. As a result, developing multi-user or publicly available solutions that do not require direct physical contact is preferable. Therefore, using biometrics can be a more hygienic alternative for authentication and access control.

1.1.3 Under GDPR

However, there are some implications regarding working with, transferring, and storing biometrics. With the widespread adoption of biometrics, privacy, and data protection concerns have arisen, particularly in the General Data Protection Regulation (GDPR[7]) context. Biometric data is considered sensitive personal data under GDPR, and organizations must adhere to strict guidelines when collecting, processing, and storing such information. This includes obtaining explicit consent from individuals, implementing robust data security measures, and ensuring data minimization and transparency in processing.

1.1.4 Enforcement of GDPR

Since May 2018 [12], the General Data Protection Regulation (GDPR) came into effect, replacing the Data Protection Directive 95/46/EC. The enforcement of GDPR has been relatively active, with numerous fines and penalties being imposed on non-compliant organizations. The European Data Protection Board (EDPB) has been

the primary regulatory body responsible for enforcing GDPR. It has issued several guidelines and recommendations to help organizations comply with the regulation. Since then, one of the most significant changes in enforcing GDPR has been increased fines for non-compliance. Initially, the maximum fine for non-compliance was €20 million or 4% of the organization's global annual turnover, whichever was higher. However, this has since been increased to €50 million or 4% of the global annual turnover.

1.2 Problem formulation

To cope with evolving market challenges, we need to be able to develop solutions flexible/modular enough to be ready to improve and adjust to new requirements. Therefore, on the example of an FR-based smart home lock, we propose to design & implement a biometrics-based security system with GDPR compliance in mind.

Chapter 2

Related Works

2.1 Detection

In computer vision, object detection is a common problem that involves identifying and localizing objects of interest within an image or a video stream. The main goal it to predict location of object of interest,i.e., bounding box within the image.

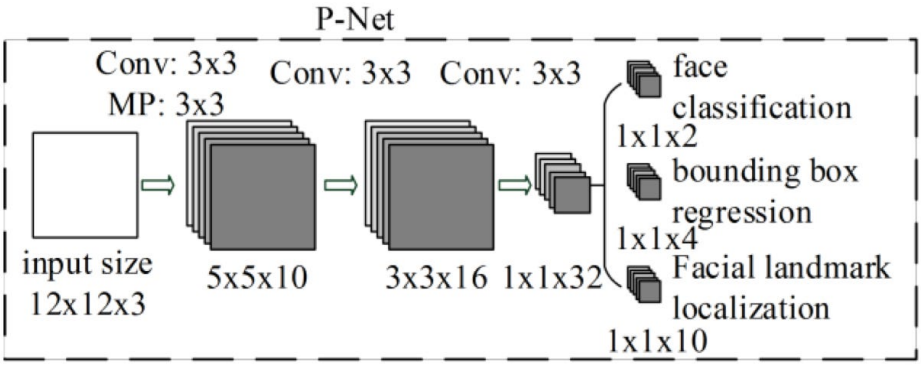
Often, bounding boxes are represented by coordinates of the top-left corner (x_{min}, y_{min}) and coordinates of the bottom-right corner (x_{max}, y_{max}) or box height and width.

2.1.1 Face Detection

In the most simple settings Face Detection requires predicting bounding boxes per each face on an image or a video stream.

Most common approaches:

1. The Viola-Jones Object Detection Framework[21]. First real-time object detection framework, and it has been widely used for face detection. It uses Haar-like features and a cascade of classifiers (using AdaBoost) to detect faces in an image. The algorithm has four stages: Haar Feature Selection, Creating an Integral Image, Adaboost Training, and Cascading Classifiers.



2. Deep Learning Based Approaches. Deep learning methods have shown excellent performance in face detection in recent years. They use convolutional neural networks (CNNs) to scan images in a multi-scale and sliding window fashion. Notable architectures include Multi-task Cascaded Convolutional Networks (MTCNN[24]) and HR-ERNet. MTCNN, for example, uses a cascaded architecture with three stages of increasingly more complex networks to propose candidate bounding boxes and refine their positions.

3. YOLO (You Only Look Once)[17]. YOLO is a popular real-time object detection system that can be trained to detect human faces. It frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities.

2.1.2 Hand Detection

1. Single Shot MultiBox Detector (SSD)[13]. The SSD framework is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes. The main advantage of SSD is its speed - it's much faster than methods that involve generating proposal regions due to the fact that it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. In the context of hand detection, the SSD model would be trained on a dataset of images with labeled hands. It would learn to recognize hands and output bounding boxes around them.

2. Hand Segmentation Using Learning-Based Prediction and Verification for Hand Sign Recognition [4]. This 1998 paper discusses a skin color-based approach to hand segmentation, which is a key step in hand detection. The authors use a combination of skin color segmentation and a verification process involving motion analysis and shape matching to detect hands in the context of sign language recognition.

2.2 Recognition

Recognition problems in computer vision involve identifying objects, persons, places, or actions in images or videos. The objective of these problems is to label or categorize the visual content according to pre-defined classes.

2.2.1 Face Recognition

Eigenfaces

Eigenfaces[20]. This is an older technique that involves performing Principal Component Analysis (PCA) on the face images to extract the most significant features, called Eigenfaces.

Deep Convolutional Neural Networks (CNNs)

This involves training a CNN to classify the face images directly. This approach can be powerful, but it often requires a large amount of labeled training data, which often is not achievable.

Deep Metric Learning

Deep metric learning involves training a deep learning model to learn an embedding space where distance corresponds to a similarity metric. In the case of face recognition, this means that the model should learn to map images of the same person's

face to nearby points and images of different people's faces to distant points.
Metrics and models:

1. FaceNet[**schroff-2015**]. It uses an Euclidean space for face representations. The model is trained using a triplet loss function. Triplet loss involves triplets of images: an anchor (a), a positive example (p) that matches the anchor, and a negative example (n) that does not match the anchor. The goal is to learn a function f such that the distance between f(a) and f(p) is smaller than the distance between f(a) and f(n). FaceNet uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer.

2. ArcFace (DeepInsight)[5]. The ArcFace method introduces an additive angular margin loss to the softmax loss (often referred to as ArcFace loss) to perform highly efficient and discriminative face recognition. It emphasizes the margin between positive and negative pairs, making the features more discriminative.

Retrieval. For real-time inference, it is crucial to optimize embedding organization and retrieval, often referred to as indexing. Common types of indexing:

1. Brute Force Indexing: In this simplest method, to retrieve the most similar objects to a given query, you would compute the distance between the query and each data point in your dataset. The main disadvantage of this method is that it doesn't scale well, as you have to compute the distance to every single data point.

2. Approximate Nearest Neighbor (ANN) Indexing: This method is a more efficient way to find the most similar objects. Instead of computing the exact distance to every data point, it uses various algorithms to find approximate nearest neighbors. The advantage is that it's much faster and can handle larger datasets. There are multiple libraries available that offer ANN indexing, like FAISS (Facebook AI Similarity Search), Annoy (Spotify), or NMSLIB (Non-Metric Space Library).

2.2.2 Gesture Recognition

1. Real-Time Hand Gesture Recognition Using Finger Segmentation[3]. This paper proposes a real-time hand gesture recognition system based on finger segmentation. It presents a method for segmenting the fingers from hand images and uses geometric features of the fingers for gesture classification.

2. Hand Gesture Recognition Using 3D Convolutional Neural Networks[15]. This paper presents a 3D convolutional neural network-based approach for hand gesture recognition. It utilizes 3D convolutional filters to capture both spatial and temporal information from the hand gesture sequences

3. Template Matching. his approach involves creating a database of hand gesture templates and comparing input gestures with these templates. Various matching techniques such as Euclidean distance or correlation coefficients are used to identify the closest match.

2.3 Split Computing

A concept where the computation process is split between the device and the cloud. In other words, part of the computation is offloaded to the cloud, which reduces the processing burden on the device and can help to conserve battery life and processing power. This is often used in conjunction with edge computing to provide a balance between processing power and latency. It is also referred to as federated learning, is a method of machine learning where the algorithm is trained across multiple decentralized edge devices or servers holding local data samples, without exchanging them. This is particularly valuable in scenarios where privacy is a concern, as it helps to maintain the confidentiality and integrity of data. Applying split computing or federated learning for face recognition tasks can provide a significant enhancement in privacy protection. In a typical centralized face recognition system, facial images or extracted features need to be uploaded to a central server for processing. This potentially raises serious privacy concerns, as sensitive personal data is being transferred and stored centrally.

Chapter 3

Methodology

3.1 Face Detection & Recognition

3.1.1 Detection

Criteria for the approach:

1. Low overhead – should allow real-time execution on an edge device, with little to no unnecessary overhead.
2. Robustness – should avoid as much as possible incorrect object detection. If a lot of incorrectly detected objects are processed and sent to the server it creates a significant overhead to the whole security system

Possible approaches:

1. Haar feature-based cascade classifier [22]. It is a lightweight classic computer vision algorithm. Its implementation is available as part of OpenCV library[16].
2. MTCNN [23]. Multi-task Cascaded Convolution Network, is robust enough DL approach. Its implementation is available as part of facenet-pytorch library [19]
3. BlazeFace[2]. It is a lightweight and well-optimized face detection model built by Google and designed for mobile and edge devices. It's part of the MediaPipe[14] framework, which is a cross-platform framework for building multimodal applied machine learning pipelines for mobile devices.

We tried both of those three approaches. As expected, the classic algorithm (from OpenCV library) has proven to be the most lightweight approach, but also the least robust one. The MTCNN has yielded the best results out of those three but had the greatest overhead which is also problematic. The BlazeFace-based face detector from mediapipe has not that great but reasonable precision and little overhead. Therefore, we have chosen the golden center between those 2 which is BlazeFace because it is both robust and lightweight enough for our hardware.

3.1.2 Recognition

Criteria for the approach:

1. Zero-shot – should not require readjustment (retraining) after adding a new user/face.

2. Robustness – should be able to determine outliers in most cases.

Due to the availability of large amounts of data [6], current SOTA FR approaches are deep network-based.
Possible approaches:

1. Transfer learning. It is a powerful technique in machine learning that allows models to transfer knowledge learned from one task to another. In the context of deep metric learning, it can be used to leverage pre-trained models, often trained on large-scale, diverse datasets, to learn useful representations in a new, possibly smaller, dataset. It can also speed up training, as starting from a pre-trained model often requires less training time to reach a good performance level.

2. From scratch. We can train a model from scratch using random initialization of the weights. This may be a viable option when our dataset is large enough and sufficiently different from the datasets on which pre-trained models are typically trained. However, this often requires a substantial amount of data and computational resources.

For this task, we have chosen Deep Metric Learning approach with the ResNet18[10] network trained using ArcFace loss [5]. The pre-trained network is provided as part of arcface-pytorch [18]) package.

Inference

We use cosine distance as the measure of similarity, the smaller the distance the more similar are two vectors.

$$d(u, v) = 1 - \frac{u \cdot v}{||u|| \cdot ||v||}$$

We assume the vectors may belong to the same user face when the distance between those two vectors is smaller than a certain threshold

$$s(u, v, sth) = d(u, v) < sth$$

sth is similarity distance threshold

We define the probability of vector q representing the face of user u, as a ratio of user u face feature vectors from the Index that are similar enough to vector q over the total number of face feature vectors of user u residing in the Index.

$$p(q, u, sth) = \frac{\sum_{i=1}^{n} s(q, u_i, sth)}{n}$$

u_i is user u face feature vector.
n is the total number of user u face feature vectors.

A lock is to be opened if and only if the person's face corresponds to one of the known users. We assume that the vector belongs to the known user if the highest user probability is also higher than a certain threshold.

$$r(q, U, sth, rth) = max(\{p(q, u, sth) | u \in U\}) > rth$$

U is the set of known users registered to a certain lock whose face feature vectors are in the Index.
rth is the recognition threshold

Split Computing

Nowadays, edge devices are gaining more and more computational resources, e.g, RPi 4B with 8GB RAM. Nevertheless, neural networks still going deeper and deeper requiring manifold computational resources than before. Therefore it is only natural to use those resources effectively. To accomplish this we can use Split Computing [11]. We split ResNet18 into two parts:

- EdgeNet – is to be used on the edge-side device, i.e., the smart lock itself. It consists of $[0, k)$ layers of ResNet18
- ServerNet – is to be used on the server-side. It consists of $[k, n]$ layers of ResNet18, where n is the last layer

Pipeline

1. Intermediate feature extraction – it occurs on the edge side. Basically, it is just transforming the detected face (128x128 RGB image) into an intermediate feature vector using EdgeNet.
2. Final feature extraction – it occurs on the server side. Here we transform the intermediate feature vector into its final form of 1024 float32 vectors.
3. Inference – by using the approach described above we determine whether the given vector belongs to the group of users registered to lock.

3.2 Hand Detection & Gesture Recognition

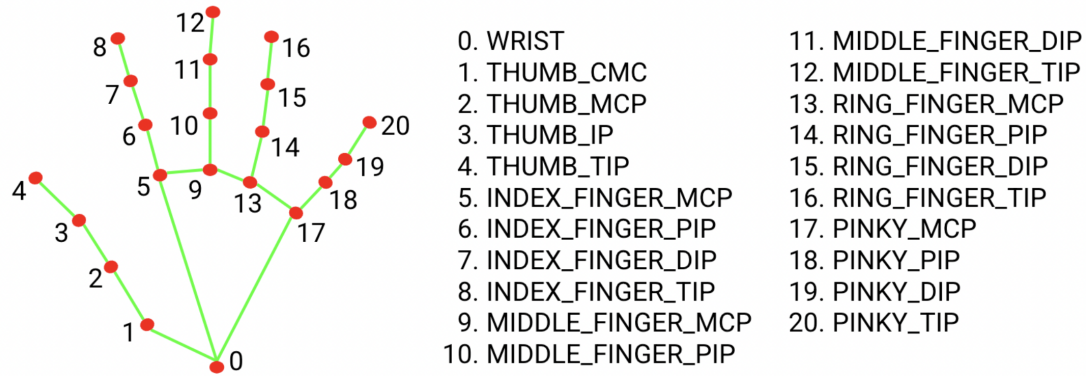
3.2.1 Detection

Criteria for the approach:

1. Lightweight – should be able to execute in real-time on the edge device.
2. Robustness – not that critical, but desirable.

For the following task, we have chosen to use the mediapipe library hands. It provides hand landmark detection capabilities and is lightweight enough to be run on an edge device.

FIGURE 3.1: Hand landmarks



3.2.2 Recognition

Having all the hand landmarks recognizing specific gestures becomes straightforward. We will use gesture recognition only for triggering the user verification process. Therefore, we need only one specific gesture, i.e., *Activation Gesture*. For that purpose, we choose *Knocking Gesture*, which can be further simplified as *Palm Open* changing to *Palm Squeezed*. We can detect those simple hand gestures using the following approach 3.1:

1. *Palm Open* – check whether all INDEX, MIDDLE, RING, FINDER finger TIPs are above corresponding MCPs

2. *Palm Squeezed* – whether all INDEX, MIDDLE, RING, FINDER finger TIPs below corresponding MCPs

3.3 Instruments

3.3.1 Programming Language

To quickly prototype and implement our system we decided to use Python as the main programming language.

3.3.2 Data Storage

To store registered user feature vectors we need some kind of database or storage. For this purpose, we decided to use object storage. It is a data storage architecture that manages data as objects, as opposed to other architectures, e.g., file systems that manage data as with a file hierarchy, and block storage that manages data like blocks within sectors and tracks. Each object in this storage system consists of data, an arbitrary amount of metadata, and an identifier that should be unique globally. Properties:

1. Flat Address Space. Object storage uses a flat address space that enables easier scaling. Unlike file systems, there's no need to organize files in a hierarchy,

which can be complex to manage and navigate as the system scales. With object storage, objects are dispersed across various devices in different locations, each accessed through its unique identifier.

2. Metadata. Object storage includes extensive metadata. In traditional file systems, metadata usually includes basic file information (e.g., creation date, modified date, size). In object storage, metadata is extensive and customizable, making the data easier to manage, analyze, and utilize.

3. Scalability. Object storage is highly scalable because of its flat address space. It's built to handle vast amounts of unstructured data, making it a good fit for large-scale cloud storage and big-data analytics.

4. Data Protection & Reliability. Object storage often includes built-in data protection mechanisms such as redundancy, erasure codes, and data replication.

5. Accessibility. Objects are typically accessible through HTTP-based APIs, making data easily accessible to web-based applications.

Object storage systems:

1. Amazon S3 (Simple Storage Service). Amazon S3 is one of the most well-known and widely used object storage services. It's designed to provide 99.999999999% (11 9's) of durability, and it stores copies of each object across multiple systems to achieve this.

2. Google Cloud Storage. Google's object storage service is similar to Amazon S3 and is designed to be highly scalable and durable. It also supports a variety of data access levels, from high-frequency access to long-term archival.

3. Microsoft Azure Blob Storage. Azure's Blob Storage is Microsoft's object storage solution for the cloud. It's optimized for storing large amounts of unstructured data, such as text or binary data.

4. IBM Cloud Object Storage. IBM's solution is designed to handle unstructured data with durability, security, and scalability, using IBM's dispersed storage technology.

   OpenStack Swift. Swift is an open-source object storage system that can scale to store petabytes of data, and it's part of the OpenStack project, which provides open-source software for building cloud services.

3.3.3 Distribution

To distribute solutions in a platform-independent way we decided to use docker images. Docker is an open-source platform that automates the deployment, scaling, and management of applications. Docker uses containerization to bundle an application and its dependencies into a single object, or "image". Docker images are lightweight, standalone, executable packages that include everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files.

Docker images are designed to be distributed and run across different platforms. This allows developers to design and test applications in a local Docker environment

and then easily deploy them elsewhere, knowing that the Docker image contains everything the application needs to run correctly.

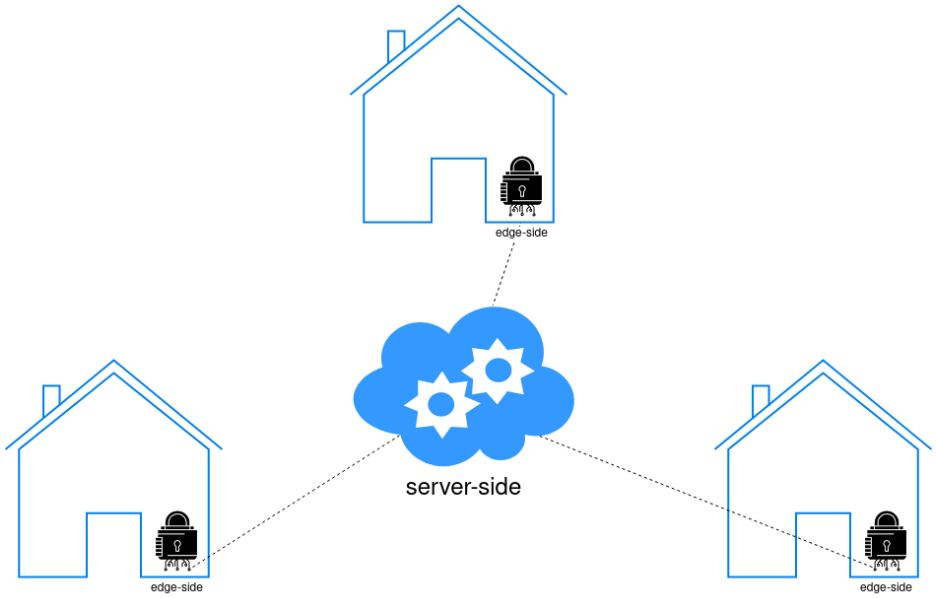
3.3.4 Deployment

To deploy our system on one instance/node we decide to use docker-compose. It is a tool that allows to define and manage multi-container Docker applications. It uses YAML files to configure an application's services and then with a single command, it creates and starts all the services from the configuration.

Chapter 4

System Architecture

FIGURE 4.1: Highlevel Architecture



4.1 Overview

To support multiple smart home locks and to be able to serve multiple authentication requests simultaneously our system is to be distributed. To The Door Lock system consists of two parts: edge-side and server-side.

1. Edge-Side
   Edge-side is the smart-lock itself. It closes/opens the door lock, takes photos, recognizes *Activation Gesture*, detects faces and does some prepossessing on them, and sends a request for user verification to the server side.

   (a) Lock

   - Raspberry Pi 4 Model B 8GB
   - Raspberry Pi Camera Module v2
   - RPi Relay Board (12V) + Solenoid Lock
   - Local UI (green/red led)

FIGURE 4.2: Lock-side connection diagram

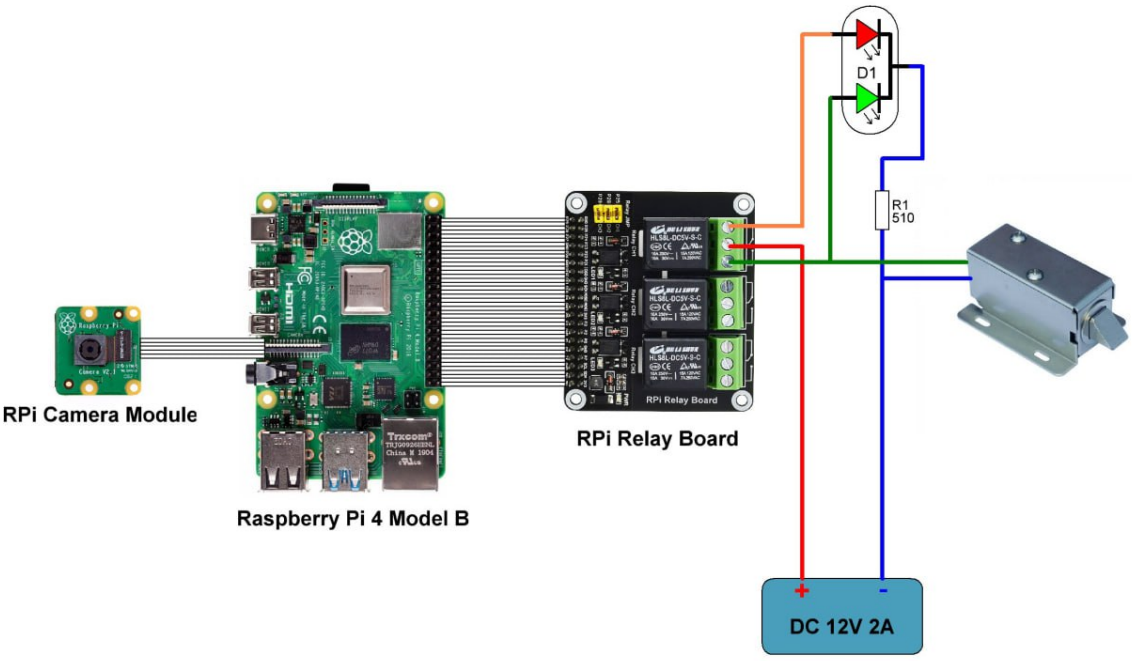
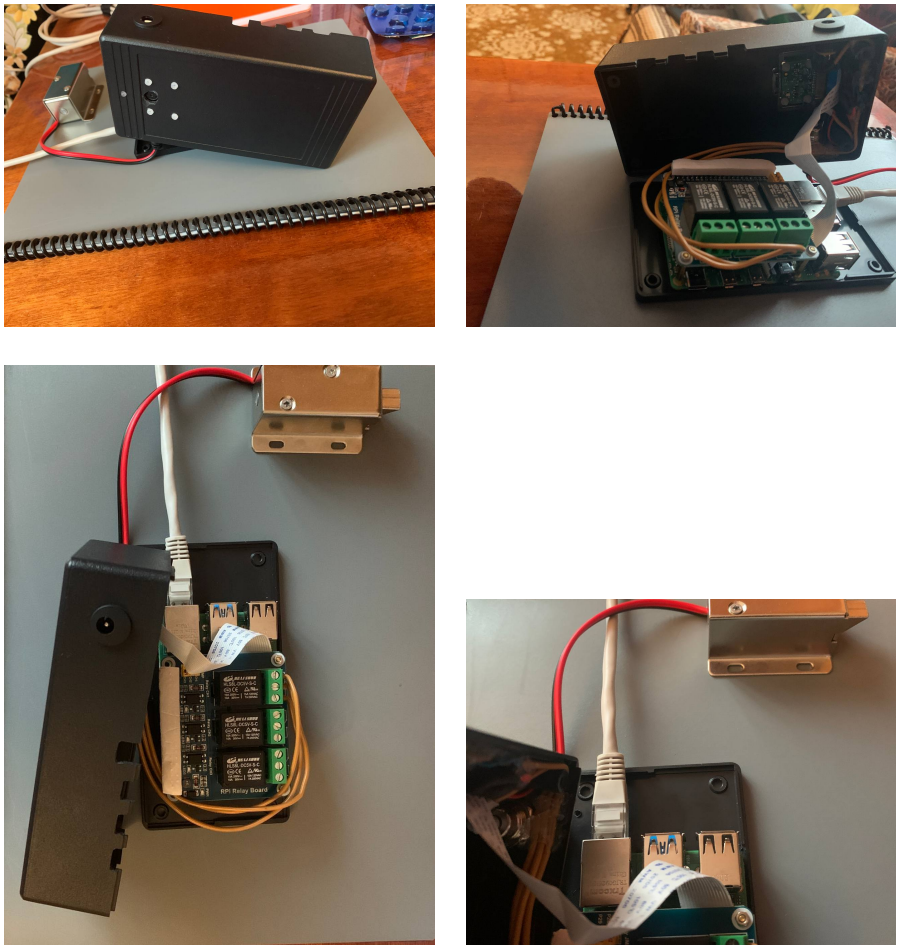


FIGURE 4.3: Lock-side physical design



2. Server-Side
    - Verification Service
    - Storage Service
    - Enrollment Service
    - Persistent storage
    - Index

4.2 Edge-Side

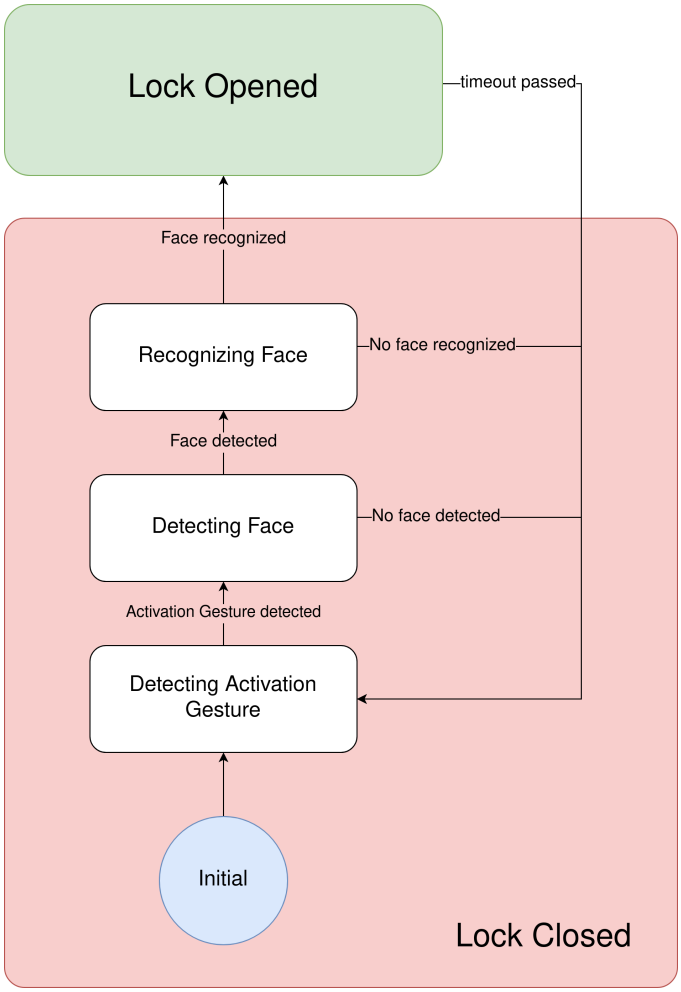
4.2.1 Lock

The core of the edge-side part of our system is Raspberry Pi 4 Model B (8GB of RAM, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GH) connected to green/red led.

The Software to be run on the edge-side is packaged into a docker image. The base image we use is *balenalib/raspberrypi4-64-debian* which is dedicated to the specific hardware. The software on the edge-side can be logically divided into 6 tasks:

- **Control Task** – This task manages all tasks mentioned below. On the lock startup, it requests for an *Activation Gesture* detection from **Activation Task**. When it receives the frames on which the *Activation Gesture* was detected it requests for face detection from **Detection Task**. If no face was detected it requests again for detection of *Activation Gesture* from **Activation Task**. If a face was detected it requests to recognize it from the **Verification Task**. If the face was not recognized it requests again for detection of *Activation Gesture* from the **Activation Task**. If the face was recognized it unlocks the door for 5 seconds and then requests for *Activation Gesture* to be detected by the **Activation Task**.
- **Camera Task** – This task streams frames taken by Pi Camera Module v2.
- **Activation Task** – This task listens to the frame stream and is responsible for detecting *Activation Gesture* and passing frames on which mentioned gesture is detected to **Control Task** on its request.
- **Detection Task** – This task is responsible for detecting bounding boxes for faces on frames provided by **Control Task** and choosing the most representative among them. The result of a successful face detection procedure is one cropped and resized to a 128x128 RGB face image.
- **Verification Task** – This task receives a 128x128 image from **Control Task** and transforms it to a 2x32768 float32 vector which is an intermediate representation of a detected face. Then it sends this intermediate feature vector to the server-side (verification service specifically) to be recognized. The result of the recognition is passed to the **Control Task**.
- **Local UI Task** – This task is optional, it only exists for the demonstration on the laptop-based edge-side device. It listens to the **Camera Task's** frame stream and displays them as a separate widget on the laptop. It also listens to responses to **Control Task** requests and displays the state of the system visually.

FIGURE 4.4: Lock state diagram



4.3 Server-Side

On the server-side part of our system, there are 3 types of micro-services that may be replicated to facilitate high-load traffic, persistent storage, and in-memory Index. Although it makes sense to have API Gateway to load balance traffic between replicas of services, there is a ready solution from each of the big cloud providers (AWS, GCP, Azure), e.g., AWS Gateway Load Balancer (GLB). So it is only natural to choose to use a ready solution from the cloud our system will be deployed to.

4.3.1 Persistent Storage

Its purpose is to store the data of all registered locks and users. In simple terms, user/lock being registered means its data being stored at the **Persistent Storage**. For this purpose, we decided to go with object storage as it is scalable and reliable and all the cloud provider offers a ready solution for it,e.g. AWS S3. For local deployment, we emulate it with the docker volume.
The vectors in this object storage have prefixes consisting of two parts:

1. Lock identifier – it specifies to which lock vector belongs, e.g., **MY-AWESOME-LOCK**.
2. User identifier – it specifies the user whose face the vector represents, e.g., **Maksym**.

Here is an example of such storage:

```
\LOCK_0\USER_0\vector_0.npy
              ...
              vector_n.npy
       ...
       USER_n\vector_0.npy
              ...
              vector_n.npy
...
\LOCK_n\USER_0\vector_0.npy
              ...
              vector_n.npy
       ...
       USER_n\vector_0.npy
              ...
              vector_n.npy
```

We can say that the lock is registered if there is at least one vector first part of which corresponds to the lock's name.

4.3.2 In-memory Index

Its purpose is to provide fast retrieval of known users' face feature vectors that are the closest to the query vector. For this, we use Redis, as it provides means to retrieve vectors based on the cosine distance (https://redis.io/docs/stack/search/reference/vectors/). Each feature vector is stored as JSON document with 3 fields:

```
vector VECTOR FLAT 6 TYPE FLOAT32 DIM 1024 DISTANCE_METRIC COSINE
user   TEXT
lock   TEXT
```

Also, each document is stored for 60 seconds in Redis, so we do not need to invalidate it manually when a new user is registered to or deleted from the lock. **Verification Service** is responsible for populating and querying Redis.

4.3.3 Verification Service

Its purpose is to serve verification requests from edge-side devices. From edge-side it receives 2x32768 float32 intermediate feature vector. Executing the second stage of the Face Recognition pipeline it transforms this vector into 1024 float32, which is used to query the **In-memory Index** and perform recognition (detect whether the vector in question belongs to the registered user of the lock). When the **In-memory Index** for the lock of interest, it retrieves lock users' feature vectors through the **Storage Service** and populates the **In-memory Index**.
It has only one REST endpoint

- Verification: using provided intermediate feature vector performs recognition and responds with the result.

4.3.4 Storage Service

Its purpose is to encapsulate access to all locks users' face feature vectors from **Persisten Storage**. It stores/deletes lock users' feature vectors provided by **Enrollment Service** into/from **Persistent Storage** and retrieves them on **Verification Service** request.
It has the following REST endpoints:

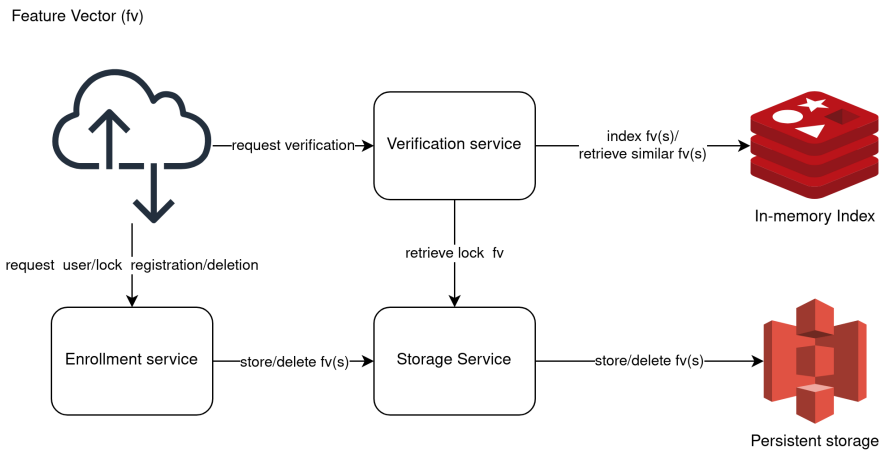
- Lock data Retrieval: retrieve feature vectors of all users associated with the lock
- Lock data deletion: delete all feature vectors of all users associated with the lock
- Lock data store: add new users/feature vectors to the lock
- Lock user data deletion: delete all feature vectors of the user associated with the lock

4.3.5 Enrollment Service

Its purpose is to serve user/lock (un)registration requests from the outside. It requests **Storage Service** to add/delete lock/user/feature vectors to/from the **Persistent Storage**. It is also responsible for running the second stage of the Face Recognition Pipeline to transform users' intermediate (2x32768 float32) face feature vectors into their final (1024 float32) form that is to be stored in the **Persistent Storage**.
It has the following REST endpoints:

- Lock user(s) registration: register user to specified lock, if lock not exists also register lock
- Lock user deletion: delete users from the associated lock
- Lock deletion: delete lock

FIGURE 4.5: Server-side architecture



Chapter 5

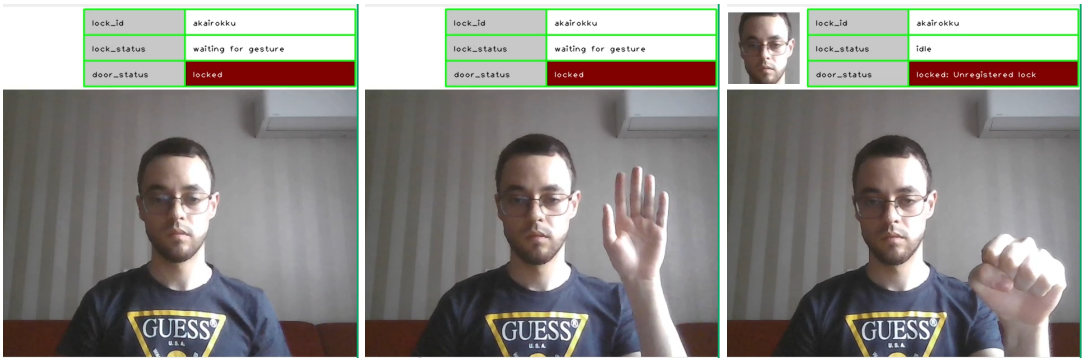
Usage

Here we demonstrate a full use-case scenario for our system. The demonstration is done on the laptop due to the lock not having any other local user interface except green/red LED.

5.1 First run

When the system is up and running for the first time there is no lock registered. Therefore, the expected behavior is that **Verification Service** will respond to edge-device requests with a status informing it that edge-device is not yet registered.

FIGURE 5.1: Not registered lock



5.2 Enrollment

To enroll lock we need to run python script. We should pass **Verification Service** URL, our lock name, enroll command, and path to the dataset of users we want to register. Sample dataset structure:

```
/path/dataset/
-------------| andriy
|              |-- 1680880160.11034.jpg
|              |-- ...
...
|------------| maksym
|              |-- 1680869428.4691794.jpg
|              |-- ...
```

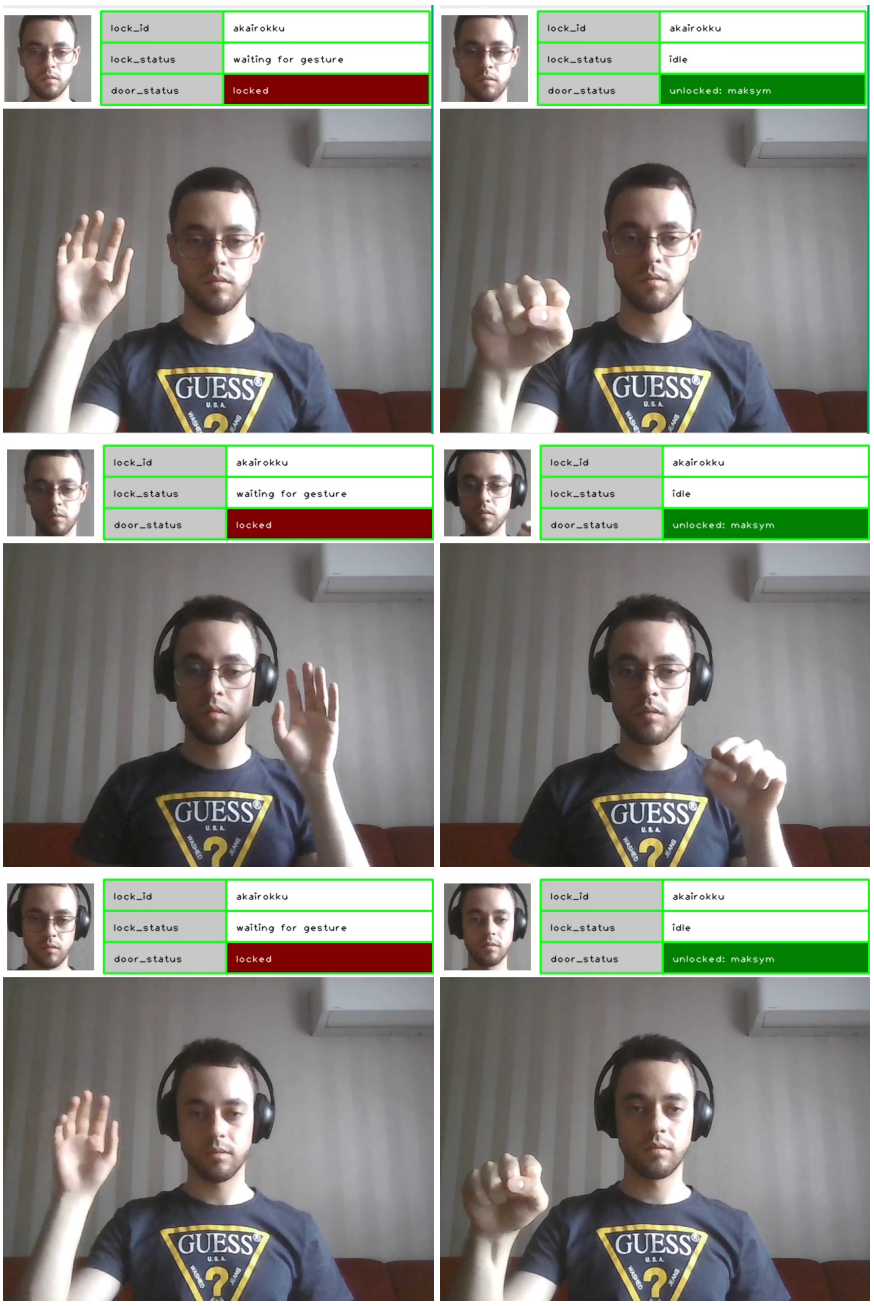
Sample command:

```
python -m enroll http://localhost:5001 my_lock_id --enroll path/to/dataset
```

5.3 Post Enrollment

After the server-side is aware of the existence of our lock and users (they are in the **Persistent Storage**) we are ready to go.

FIGURE 5.2: Registered lock



Chapter 6

Alternative Solutions

6.1 Nest Hello Doorbell

The Nest Hello Doorbell [9] is a smart video doorbell that provides a live video feed of your doorstep and offers advanced features such as facial recognition. Key Features:

- Facial Recognition: Utilizes advanced facial recognition technology to identify familiar faces and send personalized alerts to your smartphone.
- Two-Way Audio: Enables you to have conversations with visitors through the built-in microphone and speaker.

6.2 August Smart Lock Pro

August Smart Lock Pro[1] key features:

- Remote Access: With the August Smart Lock Pro, you can lock and unlock your door, control keyless access, and keep track of who comes and goes, all from your phone.
- DoorSense Technology: August's DoorSense technology helps you know if your door is securely closed and locked.
- Compatibility: It's compatible with most standard deadbolts, allowing you to keep your existing lock and keys. It simply attaches to your existing deadbolt, on the inside of your door, so you can still use your keys outside.
- Smart Home Integration: The August Smart Lock Pro is compatible with popular smart home platforms, such as Apple HomeKit, Alexa, and Google Assistant. This means you can control your lock with voice commands and integrate it with other smart devices in your home.
- Auto Lock & Unlock: The lock can automatically lock behind you and unlock as you approach, which is convenient for hands-free entry and exit.
- Activity History: You can track activity at your doorstep with a 24/7 activity feed in the August app. This means you can see when people lock and unlock your door, and when they use a virtual key.
- Easy Installation: Installation of the August Smart Lock Pro is designed to be straightforward and is touted to take about 10 minutes with just a screwdriver.

- Guest Access: You can grant access to your home for specific amounts of time to friends, family, or other people you trust. This can be done from anywhere via the August app.
- Z-Wave Plus: The August Smart Lock Pro also supports Z-Wave Plus, a wireless communication protocol used primarily for home automation.
- August Connect WiFi Bridge: The lock comes bundled with the August Connect, which plugs into a power outlet and connects the lock to your WiFi network for remote access.

6.3 Gate Labs All-in-One Video Smart Lock

The Gate Labs All-in-One Video Smart Lock [8] is a robust smart lock system that offers advanced security features along with integrated video surveillance.

- Facial Recognition: The smart lock utilizes built-in facial recognition technology to identify authorized individuals and grant them access to your home.
- Video Surveillance: The lock features a built-in camera that captures video of visitors and sends real-time alerts to your smartphone when someone is at your door.
- Two-Way Audio: It enables you to have conversations with visitors through the built-in microphone and speaker. Mobile App Control: You can control and monitor the smart lock remotely using the Gate Labs mobile app, which allows you to lock or unlock your door, view live video feeds, and manage access permissions.
- Integration: The smart lock system is compatible with virtual assistants like Amazon Alexa and Google Assistant, enabling voice control and integration with other smart devices.
- Battery Life: Due to the additional features of video surveillance and facial recognition, the lock may require regular charging or battery replacement.

6.4 Summary

There are available solutions on the market that provide better security and different ways to trigger unlock process. Nevertheless, our solution is the only one among those that uses gesture recognition for this purpose.

Chapter 7

Experiments

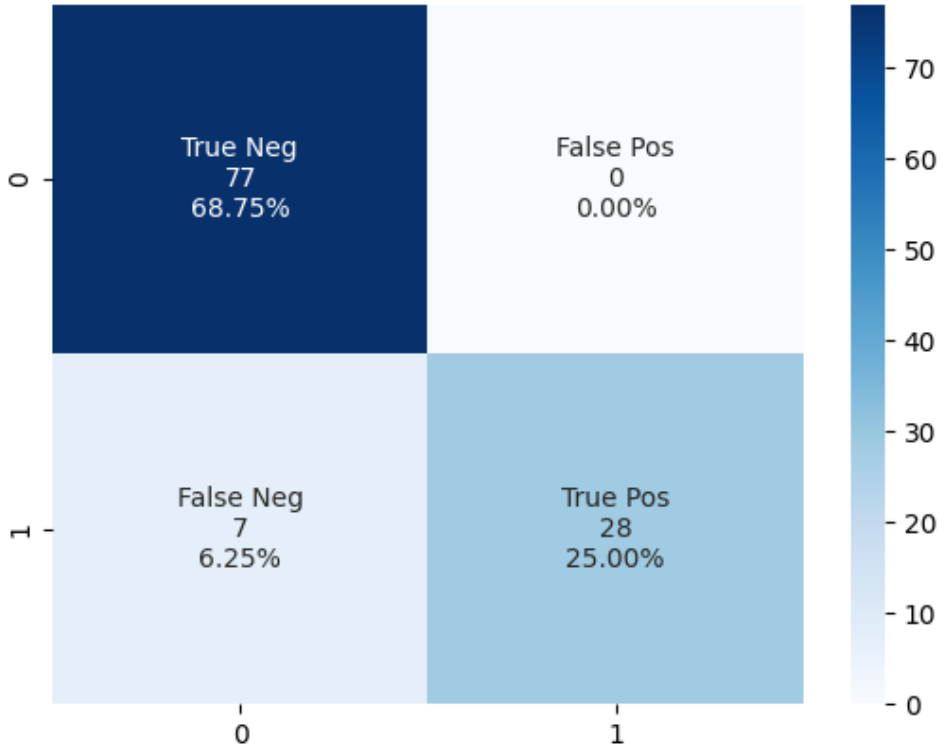
7.1 Dataset

FIGURE 7.1: Lock Dataset: Maksym B.

To test my implementation we gathered a dataset of 11 subjects, from 9 to 19 faces per subject. The dataset mainly consists of pictures of my coworkers. Therefore, we can only disclose my pictures in the table above.
We spilt this dataset into 2 equal parts:

1. Registered users
2. Unregistered users

Half of the registered users' photos were added to the system database. Each photo of an unregistered user and of registered ones (but not in the database) was verified by the algorithm. The results are the following.



As we can see there is no False Positive as it should be. Because our priority is to prevent unauthorized access. This means we need to prevent malicious agents from entering the user's home.

7.2 Latency

Obviously, on the laptop, everything runs fast and smoothly due to the abundance of computational resources which is not the case on edge devices like Raspberry Pi. Here we provide information on the latency of two most heavy tasks:

- Face Detection – on average it takes .4 seconds, up to 1 second in the worst-case scenario.
- First Stage of Face Recognition – on average it takes 2.1 seconds, up to 2.5 seconds in the works-case scenario

The latency for the response from the server-side is about 0.2 seconds on average.

Chapter 8

Summary

In conclusion, we can say that the development of the distributed FR-based door lock was successful as MVP. By any means, it is not production ready yet and has multiple limitations. The possible potential improvement includes but not limited to:

1. Liveness detection
2. Fault tolerance
3. Remote UI

Code for this project can be found @https://github.com/mak9su4roi/lokilock

Bibliography

[1] *August Smart Lock Pro*. https://august.com/products/august-smart-lock-pro-connect. [Accessed 10-Jun-2023].

[2] Dilpreet Singh Brar et al. "Face Detection for Real World Application". In: Apr. 2021. DOI: 10.1109/iciem51511.2021.9445287. URL: https://doi.org/10.1109/iciem51511.2021.9445287.

[3] Zhihua Chen et al. "Real-Time Hand Gesture Recognition Using Finger Segmentation". In: *The Scientific World Journal* 2014 (June 2014), pp. 1–9. DOI: 10.1155/2014/267872. URL: https://doi.org/10.1155/2014/267872.

[4] Yuntao Cui and John (Juyang) Weng. "Hand segmentation using learning-based prediction and verification for hand sign recognition." In: *CVPR*. IEEE Computer Society, 1996, pp. 88–93. ISBN: 0-8186-7258-7. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr1996.html#CuiW96.

[5] Jiankang Deng et al. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In: June 2019. DOI: 10.1109/cvpr.2019.00482. URL: https://doi.org/10.1109/cvpr.2019.00482.

[6] *Eight Years of Face Recognition Research: Reproducibility, Achievements and Open Issues — arxiv.org*. https://arxiv.org/abs/2208.04040. [Accessed 10-Jun-2023].

[7] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). May 4, 2016. URL: https://data.europa.eu/eli/reg/2016/679/oj (visited on 04/13/2023).

[8] *Gate All-in-One Video Smart Lock*. https://getgate.com/products/gate-all-in-one-video-smart-lock. [Accessed 10-Jun-2023].

[9] *Google Nest Doorbell (Wired)*. https://www.amazon.com/Google-Nest-Doorbell-Wired-Streaming/dp/B0B1W8WTJ2. [Accessed 10-Jun-2023].

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: June 2016. DOI: 10.1109/cvpr.2016.90. URL: https://doi.org/10.1109/cvpr.2016.90.

[11] Yiping Kang et al. "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (2017).

[12] *Legislation.gov.uk*. The National Archives. accessed 2023-04-13. URL: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted.

[13] Wei Liu et al. *SSD: Single Shot MultiBox Detector*. cite arxiv:1512.02325Comment: ECCV 2016. 2015. DOI: 10.1007/978-3-319-46448-0_2. URL: http://arxiv.org/abs/1512.02325.

[14] *mediapipe — pypi.org*. https://pypi.org/project/mediapipe/. [Accessed 10-Jun-2023].

[15] Pavlo Molchanov et al. "Hand gesture recognition with 3D convolutional neural networks". In: June 2015. DOI: 10.1109/cvprw.2015.7301342. URL: https://doi.org/10.1109/cvprw.2015.7301342.

[16] *OpenCV: Cascade Classifier — docs.opencv.org*. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. [Accessed 10-Jun-2023].

[17] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. cite arxiv:1506.02640. 2015. URL: http://arxiv.org/abs/1506.02640.

[18] ronghuaiyang@github. *arcface-pytorch*. 2018. URL: https://github.com/ronghuaiyang/arcface-pytorch,.

[19] timesler@github. *facenet-pytorch*. 2019. URL: https://github.com/timesler/facenet-pytorch.

[20] Matthew Turk and Alex Pentland. "Eigenfaces for recognition". In: *J. Cognitive Neuroscience* 3.1 (1991), pp. 71–86. DOI: http://dx.doi.org/10.1162/jocn.1991.3.1.71.

[21] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. 2001, I–511–I–518 vol.1. DOI: 10.1109/CVPR.2001.990517. URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=990517&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D990517.

[22] Paul A. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: Dec. 2001. DOI: 10.1109/cvpr.2001.990517. URL: https://doi.org/10.1109/cvpr.2001.990517.

[23] Jia Qing Xiang and Gengming Zhu. "Joint Face Detection and Facial Expression Recognition with MTCNN". In: July 2017. DOI: 10.1109/icisce.2017.95. URL: https://doi.org/10.1109/icisce.2017.95.

[24] Kaipeng Zhang et al. "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks." In: *CoRR* abs/1604.02878 (2016). URL: http://dblp.uni-trier.de/db/journals/corr/corr1604.html#ZhangZL016.