Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

PPCU Sam:
Open-source face recognition framework

BOTOS Csaba and HAKKEL Tamás
Computer Science MSc

2019

Supervisors: PhD. HORVÁTH András, PhD. OLÁH András,
PhD. REGULY István Zoltán

Emberi Erőforrások Minisztériuma

Supported by the ÚNKP-18-2 New National Excellence Program of the Ministry Of Human Capacities

Abstract

Machine learning has become ubiquitous in our daily lives as technology leaders have started to enhance the user experience with personalized content and adaptive products. Successful enterprises possess resources of data and compute power multiple orders of magnitude larger than commercially available resources. This advantage allows them to dictate trends of data-driven solutions. On the contrary, we were more interested in what can we achieve with publicly available datasets and affordable consumer grade devices using the latest available learning algorithms.

In this work we present our fully automated face recognition system, PPCU Sam. Thanks to its on-line adaptation capabilities, our face recognition system improves its performance on every single interaction with the user. We report 89.5% net acceptance rate of the recognized faces from user feedback. This mission of ours have raised so far an active user base of more than a third of our faculty and $100,000$ manually annotated images in less than 2 months of operation.

We use this work to raise awareness for the importance of machine learning, and computer vision in particular, in our daily lives. Our open-source framework is designed in a strict modular fashion, to allow extensions and improvements in different segments of our method.

Summarizing our experiences, we demonstrate how such a system should be designed from the beginning, step-by-step and compare how state of the art research solutions operate when deployed in a real-life setting.

Acknowledgement

The authors would like to give special thanks to the Sam team members at the development of face recognition system: Naszlady Márton Bese and Mitnyik Levente who designs hardware that connects our system to the existing access control system, Bartha András who helps at server administration and configuration, Steinbach László for software testing, and Gulyás Máté for help in design. We would like to say thanks to our supervisors for their patience, guidance, and constant help: Horváth András, PhD, Oláh András, PhD, Reguly Z. István, PhD.

The project was promoted several times by M5 (Hungarian TV channel), that helps our general goal with this work: call attention to the importance of machine learning and computer vision in our every day life. The project is currently hosted and supported by the Faculty of Information Technology and Bionics, Pázmány Péter Catholic University. Hakkel Tamás and Botos Csaba are funded by the New National Excellence Program scholarship. Throughout the entirety of this work we used the PyTorch [27] automatic differentiation framework.

The authors struggle to express their highest level of appreciation to Zomborszki Emese for keeping Csaba's mental health intact during the evaluation phase of this work.

Contents

1. Introduction

1.1. Aim of project

It is our main concern that recent success of machine learning is yet to be fully understood, and the field in general needs more bright minds to join. The high level goal of our project is to motivate our peers to experiment with real life applications of Deep Learning. We chose to demonstrate capabilities of neural networks on a relevant, everyday problem, for which the results are easy to interpret, attracts attention and motives further investigation: a real time-face recognition software. Core contributions of this work are the following:

- Develop reusable modular framework to collect and annotate data, and
- provide a decent baseline algorithm which can be used for comparison of future works.

Relevance Computer science has profoundly changed our life in the recent decades: Now almost everyone has a touch screen mobile phone, many smart gadgets surrounds us everywhere we go, and also administration and healthcare is aided by computers. On the other hand, the incredibly fast pace of evolution of computer architectures and software systems makes application development a challenging task because hardware and software environment up-to-date at beginning of the development can be slightly outdated by the time of final release. It is particularly true for data analysis applications where collection, preprocessing and analysis of data significantly prolongs the development time. While design and development of data analysis applications are difficult especially because of the large amount of data, that area is relatively young because smart devices capable of recording huge amount of data from crowds of users are widely available as low budget tools just from the middle of 2000s. Furthermore, data analysis is now cannot be separated from neural networks, although using neural networks is also a new approach widely adopted just in the recent years. In vast majority of cases, such a data analysis application cannot be implemented by a single developer because of the wide range of competences required, so a team is needed to be assembled and cooperate during development process.

Motivation Our teams has assembled to solve such a problem: Extend an existing access control system at our University with a data analysis system that grants access permission when face of incoming people are recognized. As both building such a data analysis system and

develop a proper face recognition algorithm are challenging, the motivation behind our work was not only to ease the process of entering and leaving the building, but also to call attention to the relevance of machine learning in our lives. Our high level goal is to provide example of a research-benchmark-deploy process, by publishing our project as a fully open-source extendable framework, and also to encourage others by giving a spectacular demonstration to start similar projects, or even extend ours.

1.2. Neural Networks

Artificial Neural Networks, been around for several decades as an alternative to classic Machine Learning algorithms, originally motivated by dynamic models of the biological brain. It was well-recognized for its ability to extract relevant patterns without manual tuning of the weights and biases of the network; however, the idea of the perceptron model has been frozen because it is proven to be unable to learn some elementary logic operations (like the XOR operation). The idea of neural networks has been frozen until the use of backpropagation (closely related to the Gauss-Newton algorithm) allowed the researchers to stack multiple perceptron layers (MLP) on the top of each other. The problem with MLPs was the time complexity, since each forward pass between successive layers required a matrix multiplication.

The appearance of GPUs allowed the MLPs to be computed in parallel on multiple cores, which was mostly trivial to implement since the computation itself consists only elementary operations and involves no decision trees. In 2012 the ImageNet Large Scale Visual Recognition Competition [29] (ILSVRC) was won by a Convolutional Neural Network (CNN), called AlexNet [19], outperforming manually designed algorithms by a large margin. The success of AlexNet resulted in the current deep learning wave, that drew attention of professionals from different fields and connected them. The variety of the applications of deep neural networks (DNN) is wide, since they are universal approximators in theory, which means if enough training samples, computation time and capacity is provided, any function can be reconstructed and modeled. With the theory aside, the solution space of applied Neural Network has its own practical boundaries.

1.3. Data Collection

On the other hand, neural networks need huge amount of curated data, and building such a dataset can be as challenging as designing a network to process it. The main difficulty of collecting data is that one has to find a solution to make both recording and annotation fully automatic because the required size of dataset can be reached only that way. Lack of diversity within the dataset can be a problem, and also structuring, storing, maintaining, and searching data is not a trivial task. Additionally, the dataset to be learned by the algorithm is also changing in our case, as the system has to be able to learn faces of new users and even faces of registered users might slightly change. To achieve this, our data collection system must be fast and robust, and the face recognition algorithm have to be adaptable.

1.4. Circumstances

Potentials Besides the extensive work done on the field of computer vision and particularly face recognition, our case has some additional potentials that help our project to succeed. First, as we are not experts of access control systems, the existence of a working access control system at our University helped a lot to start our project. Also the number of people coming daily to our Faculty (∼300-400) is optimal, as it is high enough to provide a dataset that is sufficiently diverse and large, but not too high (in case much higher number of users it would be really difficult to store and maintain data and also much higher performance devices would be needed to record data). Finally, we were lucky to have such a supporting community of users who were open for new things and helped us by manually annotate their data.

Challenges During the development of the system, we also encountered several challenging tasks:

1. While it is an advantage that we could connect our system to an already existing professional access control system, it also challenged us because our solutions were constrainted by that system, and our extension had to be totally separated, i.e. even if our system fails, it is not allowed to break the existing system meantime.
2. The system must be operated continuously, so it was not easy to test and upgrade.
3. We had to be careful to ensure confidential usage of personal data we collect.
4. Dataset is changing over time both because of the changing lightning conditions, and changes of faces, as well.
5. We provide a personal user interface on our website, so that website must be well optimized, fast, and compatible with large range of mobile phones as well as desktop machines.

1.5. Outline

Our core contribution lies in describing our modular framework that provides an excellent set of benchmarks for the state of the art face-recognition algorithms.

In **Chapter** 2, we give a complete overview of the different modules that constitutes the Sam face recognition software: the original access control system, the data collection details, the user interfaces and the deployment of the face recognition algorithm on distributed systems.

In **Chapter** 3, we list the methods which we considered related to our goal, and provide an extensive overview of recently published face recognition datasets. Next, we describe the core recognition algorithm of our framework from theoretical aspects: the single-frame identification process that generates an ID token for the user, the multi-frame verification algorithm that utilizes the continuous video stream. We detail the dynamically adapting search database that

lets our users train the algorithm in real time. To provide comparable baseline results for future work, we implement and train various cutting edge face identification algorithms using external public datasets and evaluate their performance on our data.

Finally, in **Chapter** 4, we summarize our experiments, how we provided solution for the challenges posed by the real world application. In a narrative tone we describe our experiences gathered throughout the research and development phases of the project. Lastly, we indicate current weaknesses of our solution and suggest directions for future work

2. Technical implementation details

2.1. Original Access Control System

When started developing the system, we took advantage of the existing access control system installed all over of Faculty of Information Technology and Bionics at Pázmány Péter Catholic University (PPCU), and attached our hardware and software to it as an extension leaving the core system unmodified. That way we could easily test and deploy our system without having to worry about breaking the system and allowing it to function as it did beforehand.

2.1.1 Preconditions

Access Points and keys The access control system at our Faculty is based on a simple and straightforward protocol: All students and employees have a card or fob (for sake of simplicity, we will refer to both cards and fob keys as *cards* later in this work) with a built-in chip that can be read by a card reader at multiple spots (by all entrances of the building and next to doors of computer labs). These card are assigned to a single person, and can open subset of doors and entrance points based on a multilevel permission system updated regularly as the role of student or employee changes. Even though our system could be installed at multiple spots all over the building, we decided to concentrate on main entrance only.

Main entrance The main entrance is designed to be secure and also be prepared to let multitudes of people in during a short time. To fulfill that requirement, there are two turnstiles next to each other, and there is surrounded space with length of approx. 1.5m between the turnstiles and the main door. These circumstances are advantageous to install a face recognition system because

1. we could allow entering people to choose whether they want to use our system or not by installing it on only one of the two turnstiles,
2. there is a narrow and straight path where users can enter the building, so people choosing the turnstile with the camera on the top must enter the field of view of the camera, and
3. that path is long enough to record faces of users during 1-3 seconds until they reach turnstiles and check-in with their cards.

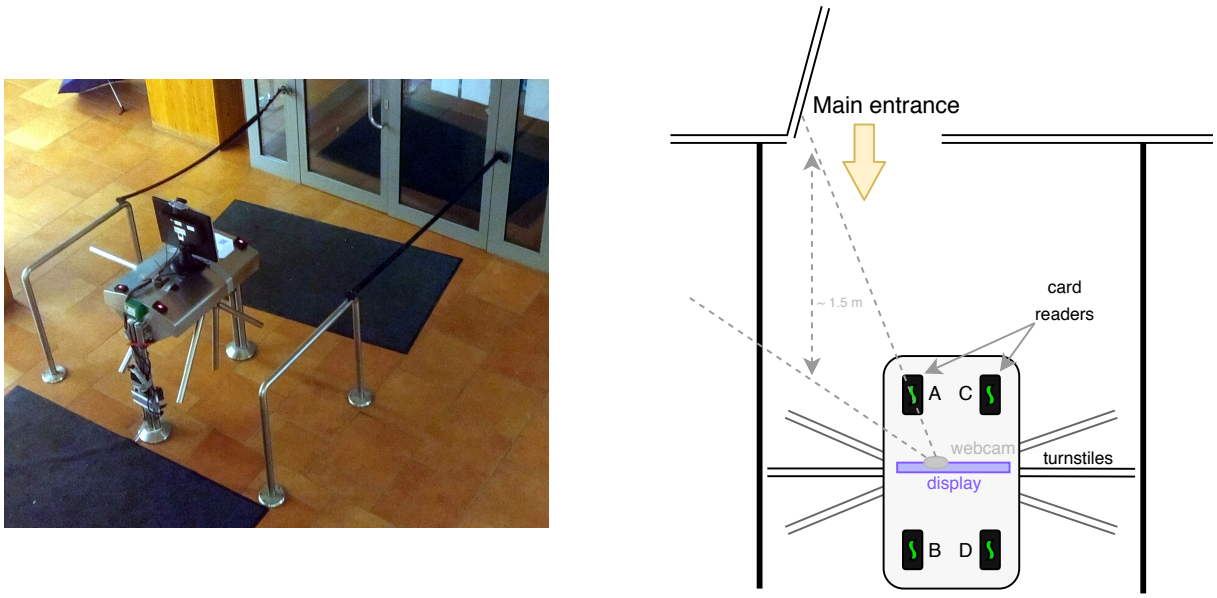


Figure 2.1: Photo and detailed plan view of the access control system at the main entrance.

Our system is working now only for incoming users. Figure 2.1 shows the main parts and parameters of the system at the main entrance.

2.1.2 Technical details

The core system consists of 3 different parts: card reader, access control server located in the server room of the Faculty, and a controller for turnstile mechanics.

Gate electronics Once a user enters the building, he or she waves his or her card over to the card reader. In our system the readers are dual-frequency readers supporting both 125 kHz and 13.56 MHz (Mifare) technology. The card contains a passive RFID (Radio Frequency Identification) chip that stores the card ID, which identifies the cardholder. The length of the ID depends on the tag: keyfobs are 125 kHz tags storing a 3-byte (24 bit) ID only, while cards are MIFARE Classic cards not only storing a 4-byte (32bit) ID, but additional 1 (or 4) kilobytes of data as well. The dual-frequency card reader reads the (3 or 4 byte) ID only, and so both cards and keyfobs are storing a 32 bit long ID (that is absolutely sufficient considering the high numbers of possible IDs: $2^{24} = 16777216$). That ID is emitted from the tag and detected by the card reader when the tag enters electromagnetic field of the reader. The card reader uses Wiegand interface that is a commonly used technology to connect a card swipe mechanism to the rest of an access control system. In our cast, it is only used to send the tag's ID to the access control server, and it consists of five wires called DATA0, DATA1, LED, BEEP, and TURN. When the reader receives the radio frequency (RF) data from the card, it translates the data to Wiegand protocol, and sends the complete binary string to the server. Zero bits transported on the DATA0, ones on the DATA1 wire. A beeping sound indicates when data is successfully sent. The server then combines data arriving on the two data wires into the original set of binary data [7].

Authorization The access control server checks permissions for the given ID, and sends answer back to the controller controlling the turnstile belongs to the card reader which sent the data. The answer is returned on LED, BEEP and TURN wires. LED and BEEP wires carry instructions to card reader when change led color to green and when to beep, respectively, and impulse on TURN wire means that turnstile has to be opened. If the returned answer is that access permission is granted than the card reader beeps again and changes the color of the led from red to green for a second, and unlocks turnstile. If the returned answer is that access permission is denied, then controller also gives feedback to the user by a long beep, but the turnstile remains locked.

2.2. Our Extension

2.2.1 Overview

To accomplish our task we had to solve multiple task on multiple platforms. First, we have to collect raw data: record and save entries and camera pictures. Second, we need to register users and connect raw data with them. Third, we need to monitor incoming video stream to recognize faces of registered users. Finally, we need to make decisions (whether or not open the gate) and visualize output. Our system operates on 5 platforms:

- Microelectronics: Hardware that listens to the signals of the card reader and capable of emulating its signals, and also a microcontroller that connects hardware to Edge.
- Edge: A high performance embedded system located in the gate that serves as a relay center: 1) Receives data from Hardware and the web camera, 2) saves data from hardware to Database, 3) stream web camera photos to Cloud, 4) display photo stream coming back from Cloud, and 5) forward instructions coming from Cloud to Hardware.
- Cloud: A GPU server that processes images to find photos faces, match them to the ones already assigned to registered users, and make decision whether or not grant access permission.
- Website: A mobile friendly website that does the registration of new users, allow users to edit or delete their personal data, and provide an interface to annotate images assigned to the given user.
- Database: Stores and retrieves temporary and persistent data: Photos, entry logs, user data, and annotations.

Figure 2.2 displays connectivity between these platforms. Our team has 4 developers: MITNYIK Levente and NASZLADY Márton Bese is responsible for the Hardware part, BOTOS Csaba expanded code basis of Edge and Cloud, and HAKKEL Tamás designed and developed the Website and Database.

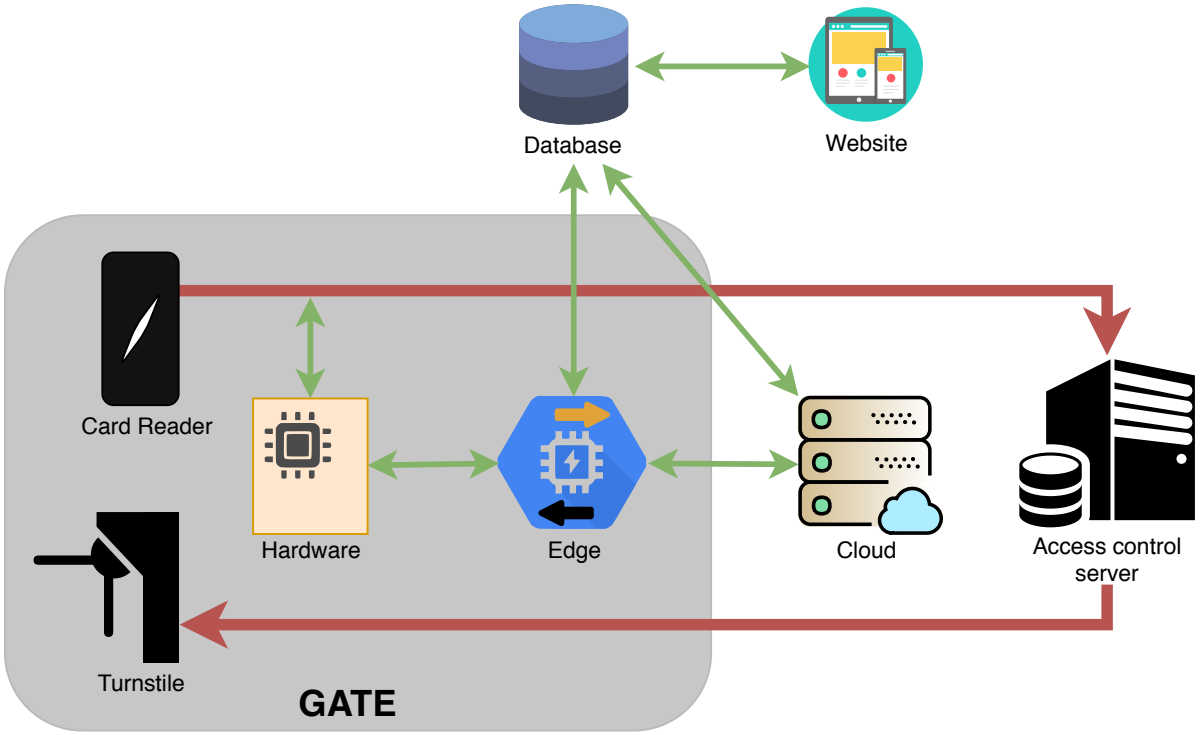


Figure 2.2: Overview of our extension, and connections between platforms.

2.2.2 Microelectronics

On the Hardware level there is three tasks to be solved: 1) listen to signals on Wiegand interface coming from card reader, 2) emulate signals of reader, and send them to access control server, and 3) communicate with Edge. The Hardware level is consists of two boards: An STM32F103C8 32-bit AVR microcontroller with auxiliary components (such as external clock and power regulator), a custom board designed to inspect and control signals of the Wiegand interface, and a symmetric power supply to produce proper voltage to connect to Wiegand bus. Figure 2.3 shows a photo these boards. Fig. 2.4 shows an overview of the custom board, and see hardware schematic in appendix A. for more details of custom board.

Transparency and separation The main feature of the custom board is that it has to be totally transparent from the viewpoint of the card reader and access control server that decodes Wiegand signals, i.e. it will only tap the communication line, so will not disconnect card reader and server even in case of a failure. The line can only be broken off for a short time when the microcontroller wants to send a card ID to the server, and wants to avoid interference between card reader and emulated signal. That requirements are satisfied by placing two N–channel FET amplifier (these are depletion transistors open at zero and positive gate voltage, and closes when negative voltage applied), and two Schottky diodes preventing current flowing back towards FET transistors. These FET transistors are driven by an opto-couplers that receives driving signal from the microcontroller through driver logic.

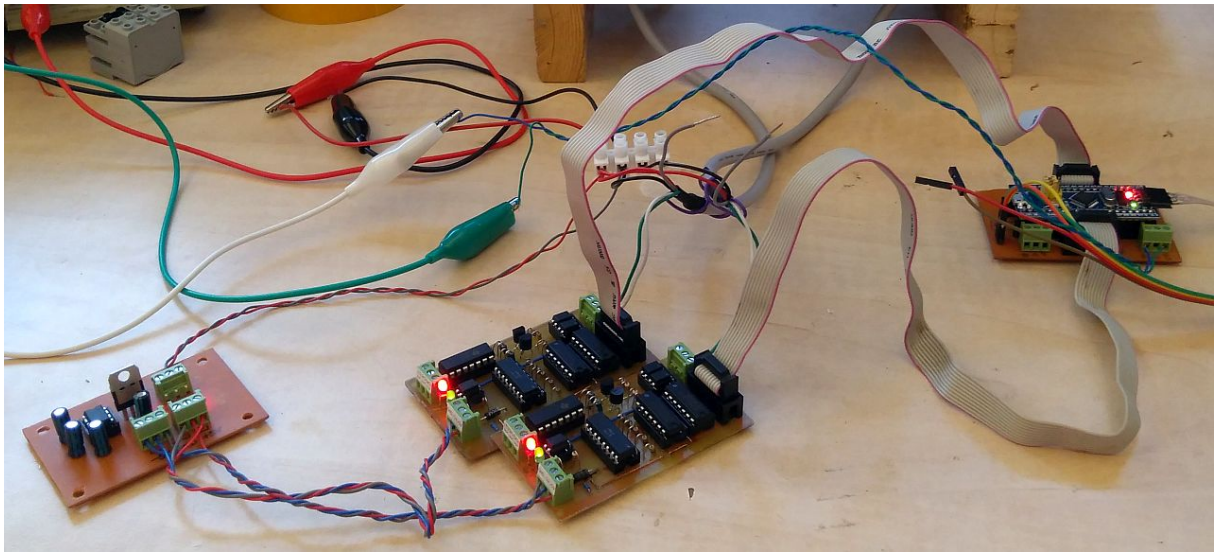


Figure 2.3: Symmetric power supply (left), two custom boards (middle) and microcontroller (right). The symmetric power supply gets energy from the same source as the card readers and feeds 0V, +5V and -5V to the custom boards. Custom boards listens and injects card IDs to Wiegand buses (blue and white wires entering custom boards from the right). Microcontroller are connected to both custom boards. It relays data between the custom board and Edge and also encodes and decodes card IDs going to and coming from Wiegand interface.

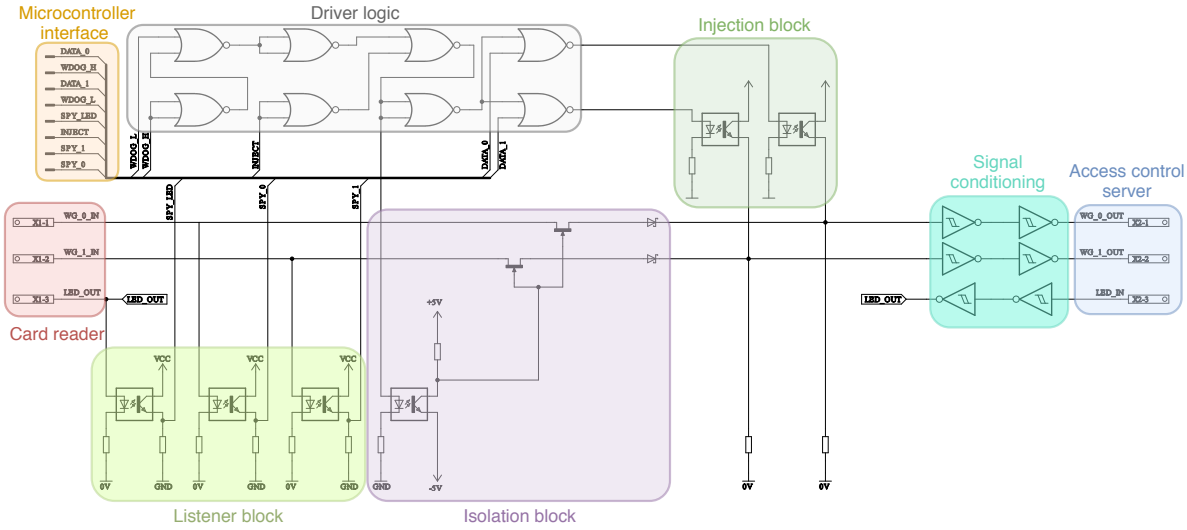


Figure 2.4: Overview of the main parts of the custom board. Driver logic instructs insulation block to separate card reader from the access control server and allows injection block to send data to Wiegand bus, but only when appropriate enabling signals are coming from the microcontroller. Listener block forwards signals coming from card reader to microcontroller. Signal conditioning components compensates effect of listener, insulation and injection blocks.

Drive logic and injection The role of driver logic is allow injection to the Wiegand interface only when appropriate enabling signals are coming from the microcontroller. Five pins of the microcontroller is responsible for injection: DATA_0 and DATA_1 carries bits to be injected to the two Wiegand wires, WDOG_H and WDOG_L are outputs of watchdog timer that ensures the microcontroller is running properly (and restarts microcontroller when something goes wrong), and the INJECT pin carries the enable bit for injection. When WDOG_H is high, WDOG_L is low, and INJECT is high, then insulator switch breaks the line between card reader and built-in controller, and drives emulation switches to allow DATA_0 and DATA_1 be sent to the controller. Driver logic is implemented on two quad 2-Input NOR gates, and emulation switches are also opto-couplers.

Listening and isolation The microcontroller also has pins to receive data from Wiegand interface and LED in the card reader (that shows the information whether access permission for the given user is granted or not). These pins are connected to Wiegand wires through opto-couplers, as well. The reason, why we used opto-couplers when connecting the microcontroller and Wiegand bus is that we wanted to isolate these two circuits galvanically to prevent connecting the two independent power sources that can cause many problems (noise due to ground loop). These two systems (microcontroller and the original system including Wiegand interface) uses two isolated power supplies, labeled with (GND, VCC) and (0V, -5V, +5V) on schematics (fig. 2.4 and appendix A). GND-VCC is the power supply of the microcontroller, and +5V and -5V power sources are generated by a symmetric power supply circuit with a ground (0V) equal to the ground of Wiegand interface. We needed -5V only for the FET transistors because we need negative voltage to close these transistors. Additionally, to compensate the effect of listener and isolation switches, signal conditioning components is inserted into the Wiegand bus.

Microcontroller - listening To listen to Wiegand wires and inject card IDs to it, and communicate with the Edge, we used a STM32 medium-density performance microcontroller that incorporates a high-performance ARM Cortex-M3 32-bit RISC core. That microcontroller uses timers to listen to data coming from card reader. One is used to sample the signal on the Wiegand wires in every $32\,\mu s$. Value read from the wire is loaded into a circular buffer that is used to oversample an incoming signal in order to provide some noise filtering capability. Circular buffer operates with tri-state logic; if all the values are the same the current state is set to that value otherwise to X (non-deterministic) state. The length of that buffer is 4 bits because the width of an impulse in the Wiegand format is at least $160\,\mu s$ ($500\,\mu s$ at most), so by the sampling rate of $32\,\mu s$ we can sample the signal at least 4 times during the impulse. There is also an another buffer that stores the bits read from the Wiegand interface. The size of that buffer is 34 bit because the card reader installed in the gate uses the standard 34-bit Wiegand data format that consist of 2 parity bits (the first and the last bits), 8 bits for faculty codes, and 24 bits for card number. We are now interested only in the card number, so faculty code is discarded. Whenever a falling edge is detected on either lines, a true of false value (it depends on which wire the impulse is detected) is appended to that buffer. To observe the end of the data stream,

another timer is scheduled to interrupt in every $8192\,\mu s$. That interval is sufficient because it is longer than the transmission time of a bit (delay between bit transmissions in Wiegand protocol is 2200-$4840\,\mu s$). When all the 34 bit arrived, than the content of the buffer (the card ID) and the source of data (which card reader emitted that bit string) is send to the Edge via serial communication using the UART (Universal asynchronous receiver-transmitter) circuit of the microcontroller.

Microcontroller - emulation Besides listening to signals, the our system is also able to inject signal to the Wiegand wires emulating the signal of the card reader when a command arrives from the Edge to do so. That command is also arrives on serial port through the UART interface and contains the card ID to be injected. The microcontroller listens to that command by setting an interrupt for that input, and when all the bits arrives of the card ID, then it pulls up the INJECT pin breaking the wire between card reader and access control server, and starts to send received card ID to server via DATA_0 and DATA_1 pins.

Hardware design and source codes are available at
https://dev.itk.ppke.hu/sam/gatepirate.git.

2.2.3 Edge

The Edge is a high performance embedded system located in the gate that serves as a relay center. We used a NVIDIA Jetson TX2 Module because it i a power-efficient embedded AI computing device which is built around an NVIDIA Pascal^{TM}-family GPU and loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth. We connected that device to Hardware via serial port through USB port (so we needed a serial-USB converter), and via web sockets through ethernet cable to the local network of the University to access internal servers of PPCU. As operating system, we installed a full Linux kernel and the Ubuntu distribution over it. We used Python3, and our scripts imports the following libraries (besides the standard libraries): numpy (1.15.1), opencv-python (3.4.2.17), pyzmq (17.1.2), tornado (5.1), zmq (0.0.0).

Processes During normal operation, there are three processes kept running in the background: *mainscreen*, *tunnel*, and *card_log*. To keep them running and restart automatically when computer restarts or program is aborted, we used crontab and screen programs. Process *tunnel* is an SSH tunnel which works by setting up an ssh connection between two hosts and use that connection to transport normal network traffic. On one side of the tunnel, the OpenSSH software takes anything that is sent to a specific port and sends it over to the other side of the connection. When the packets arrive on the target side, the OpenSSH software forwards them to the correct local port. Process *card_log* is responsible for communicating with the Hardware: First it receives card swipe events via serial port, saves them to database, and also forwards that event to *mainscreen* process through a simple file. Second, it listens for gate-opening commands coming from *mainscreen* through another file, as well. Finally, *mainscreen* process orchestrates multiple tasks:

- It reads image from web camera,
- monitors file where *card_log* process forwards card swipe events,
- send image recorded by web camera to Cloud through *tunnel* along with card swipe event if detected one,
- receives answer from Cloud containing the following information:
  - bounding box (a rectangle over the closest detected face) on the image sent, if present,
  - a dictionary/map to connect card IDs and user IDs of registered users,
  - a command to open gate, if a face is recognized,
  - list of predicted users based on the recorded image (user IDs of users whose faces are the most similar to the one detected currently),
  - final suggestion of the face recognition service,
  - tracker ID for tracking moving faces, and
  - consecutive occurrence: how many times the given face is detected in the last couple seconds
- draws information window displayed on the screen facing the user (that window shows the frame captured from the camera, frames the detected face if present any, and prints a short list of user IDs belonging to the most similar faces and also a number that represents the strength of the strength of prediction for each of that list), and
- instructs the Hardware over *card_log* process to open the gate if any registered user is detected on the captured fame by emulating their card IDs.

Source code of scripts running on Edge are available at `https://github.com/botcs/sam`, and *mainscreen* and *card_log* processes can be lauched by running *client.py* and *serialmonitor.py* scripts. Fig. 2.5 shows a photo of the user interface of the Edge displaying the video stream captured, detected faces circled, and most probable predictions listed.

2.2.4 Cloud

Cloud is a high performance GPU server, in our case Nvidia-K80 located at PPCU. It communicates with both Edge and Database via web sockets. We have written our code also in Python3, and our scripts have the following dependencies (besides the standard libraries): cffi (1.11.5), dlib (19.15.0), future (0.16.0), numpy (1.15.1), opencv-python (3.4.2.17), Pillow (5.2.0), pycparser (2.18), PyMySQL (0.9.2), PyYAML (3.13), serial (0.0.67), six (1.11.0), torch (0.4.1), torchvision (0.2.1).

Server script We have only one script kept in background in a screen session, and that script is doing all the communication with both Edge and Database, and menages face recognition. The main steps of the script:

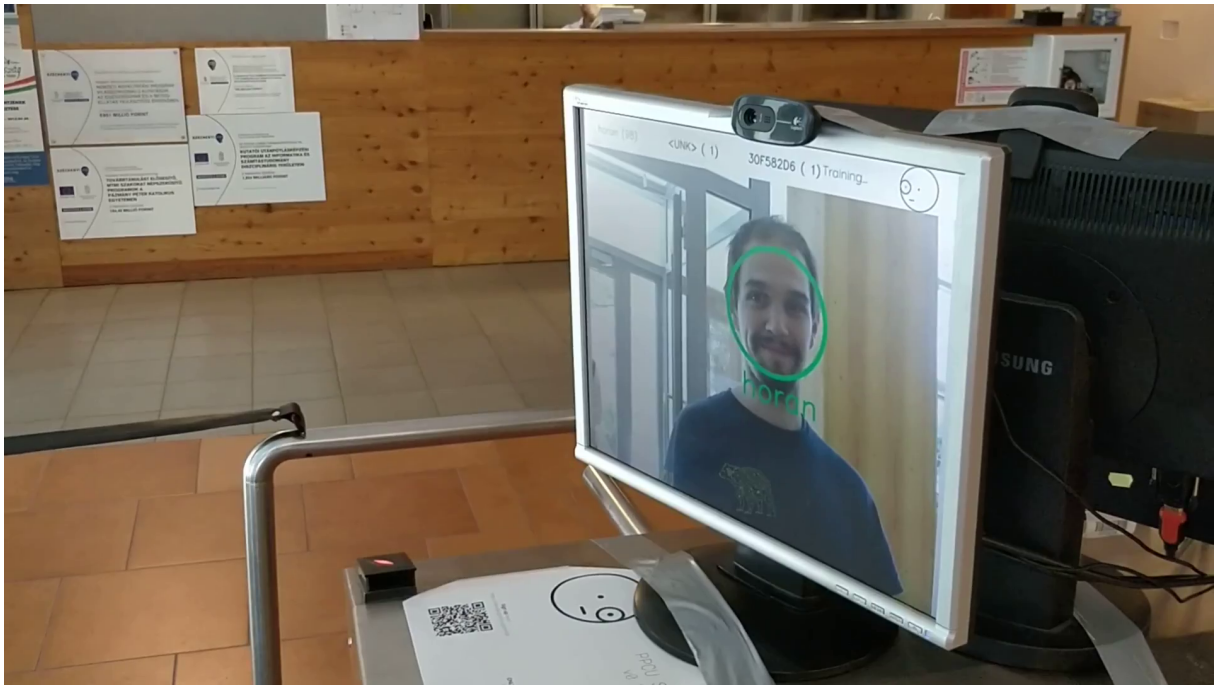


Figure 2.5: Photo of user interface of the Edge. On the screen, one can see the video stream recorded by the camera, but detected faces are circled, and by the top of the screen list of most probable predictions are listed as well as the strength of prediction.

- Read captured image and card events from Edge,
- asynchronously update of *card2name* dictionary/map that stores card ID - shibboleth ID pairs periodically (we need to update it frequently [e.g. in every 50th iteration of event loop] because we want to display shibboleth ID on screen of Edge even right after registration),
- find faces on images using dlib-based alignment module, and select the one with the largest bounding box,
- pre-process image (convert to proper format, crop face, and align face to have no angle),
- embed image (project high dimensional raw image to low dimensional representation space),
- compare to existing embeddings (kind k-nearest neighbor, and thus the k most similar images),
- track faces (once the ID is revealed of the traced box (i.e. by reading the card), then all previous bounding box are tagged as well retrospectively, when card authentication or face recognition occurs then assign latest frame ONLINE, and when tracing session dies [i.e. x coordinate of bounding box jumps further than threshold], then assign all previously unassigned),
- send data back to Edge.

Although we store images and user data in the Database, we needed to download a sub-

set of data for performance reasons. We download the latest images for all users and pre-calculated embedding vectors for them. For more details see 3, or *server.py* available at https://github.com/botcs/sam/.

2.2.5 Website

We also created a user interface besides the one of the Edge, a website to accomplish the following tasks:

1. Register new users,
2. connect users and their card IDs,
3. display recorded photos and stored personal data, and
4. provide interface to annotate photos.

The website is hosted on the servers of the University, and designed to function well on both mobile and desktop PC.

Shibboleth authentication As we planned to deal with only the students and employees of the University, we were able to use the already existing web authentication system of the University which involves all students and employees. That system is based on Shibboleth that is an open source and freely available single log-in system for computer networks and the Internet which allows people to sign in using just one identity to various systems run by federations of different organizations or institutions. In our case that authentication system is responsible to authenticate users on various web services of PPCU and it is also used to log-in to computers in computer labs. It is both secure and gives necessary information about the logged in user, so we decided to use it to authenticate visitors of the website. It is easy to set-up on any website hosted by the PPCU: Only a couple commands have to be appended to the .htaccess files located in the root of the restricted folder-system, and the system already works. Then the following protocol is executed whenever a http request arrives to the server:

1. When the user attempt to access the protected resource, the resource monitor determines whether they have an active session. If they do not, then directs them to the service provider in order to start the SSO (Single Sign-on) process.
2. The user arrives at the Service Provider (SP) which prepares an authentication request and sends both the request and the user to the Identity Provider (IdP). The Service Provider software installed on the same server as the resource, while Identity Provider are on a securely separated server.
3. When the user arrives at the Identity Provider, it checks to see if the user has an existing session. If they do (e.g. because they are already signed in to another web service that uses Shibboleth), they proceed to the next step. If not, the Identity Provider authenticates them (e.g. by prompting for, and checking, a username and password) and the user proceeds to the next step.

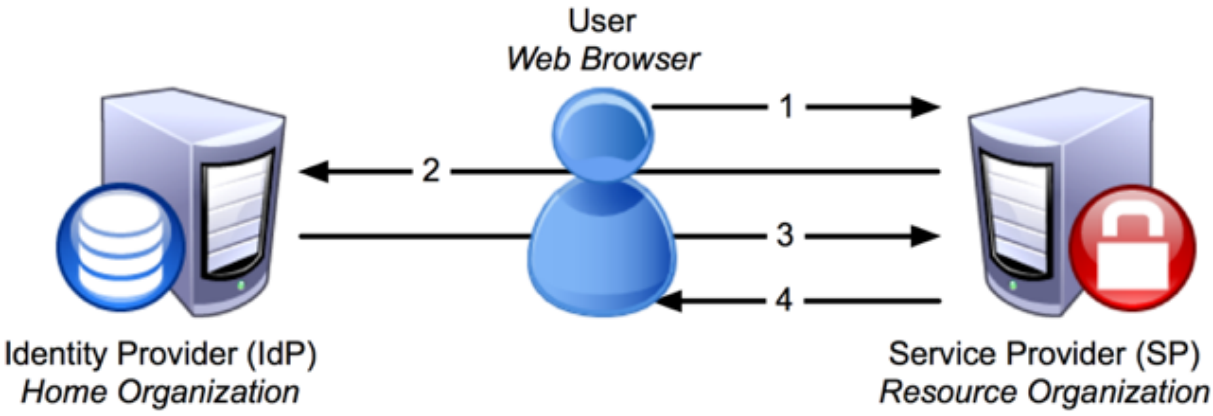


Figure 2.6: Process of authentication using Shibboleth: "1) The SP detects the user attempting to access restricted content within the resource. 2) The SP generates an authentication request, then sends the request, and the user, to the user's IdP. 3) The IdP authenticates the user, then sends the authentication response, and the user, back to the SP. 4) The SP verifies the IdP's response and sends the request through to the resource which returns the originally requested content." Image and description is adapted from wiki.shibboleth.net [3].

4. After identifying the user, the Identity Provider prepares an authentication response and sends it and the user back to the Service Provider.

5. When the user arrives with the response from the Identity Provider, the Service Provider will validate the response, create a session for the user, and make some information retrieved from the response (e.g. user's identifier, name, and email address) available to the protected resource. After this, the user is sent to the resource.

6. As in Step 1, the user is now trying again to access the protected resource, but this time the user has a session and the resource knows who they are. With this information the resource will service the user's request and send back the requested data [1].

The process of redirecting users between services are depicted on figure 2.6. The same thing from the viewpoint of the users: They simply see that they are redirected to the IdP site, and either redirected back to the requested page immediately when they have an active session, or redirected back after entering user credentials.

Registration and card authentication Using the Shibboleth authentication, the registration process is simplified in a large extent because the only thing the users have to do is approve storing their personal data (shibboleth identifier, name and email address provided by Shibboleth IdP, and photos taken by the camera on the gate) after reading a disclaimer about that, and connect their card to their user profile. The latter is done by the cooperation of Hardware and Website: When users started card authentication process on the website, it prompts the user to check-in with their card three times, and starts to monitor access log provided by Hardware. When 3 check-in events is logged from the same card ID within 20 seconds, then it connects that card ID to the user who initiated card authentication. Card authentication can also be done later, separately from registration (e.g. when user gets a new card). To secure the process, card authentication is restricted to the time window of 20 seconds starting from the initiating

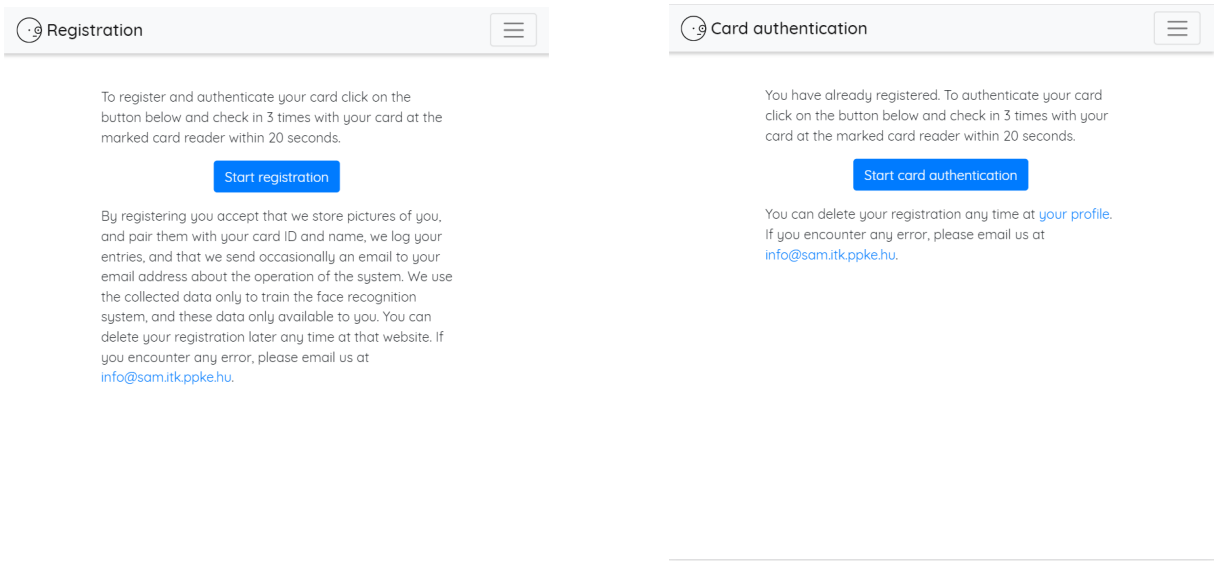


Figure 2.7: Registration (left) and card authentication (right) pages. On the registration page, user can complete registration only by clicking on the button, and all necessary data is fetched from Shibboleth IdP. Card authentication page is used to connect card IDs to users. Card authentication process can be initiated by clicking on the button, and then the user has to check-in three times to complete card authentication.

button-click on the website, and only 5 trials are allowed a day. The reason behind these restrictions are the following two use cases of possible maleficent users: 1) If we allow multiple card authentication to happen in the same time, a user might "steal" other users' card ID by keeping renewing the authentication process during the whole day, and thus the "hacker" might catch the check-in events of access log before the real registering user. 2) If we does not allow multiple users to initiate card authentication at the same time, a user blocks card authentication of other users also by continuously renewing authentication process instantly after timeout. Fig. 2.7 shows screenshots of mobile view of registration and card authentication pages.

Display photos The other main feature of website provided for registered users only is displaying recorded photos, and providing a user interface to annotate them (whether these photos are showing the given user or not). The photos are divided into two groups: Verified images are recorded when the user used their card to get access, and unverified images are recorded when the system detected a face that is similar enough to the user' verified images to trigger gate opening. These images are displayed separately on two pages accordingly. The design of these two pages are almost the same: These pages display thumbnails of photos (square crop around the face) grouped by time and location of face on the original image. Photos are listed in decreasing order by time (latest first), and older images are loaded with lazy load (loaded only when user scrolls to the bottom of the page). Right click (desktop PC) or long tap (mobile) reveals the full size photo.

Annotation The difference between the interface of the two pages lies in annotation features. While unverified images can be classified into two groups, ones to be accepted, and others to

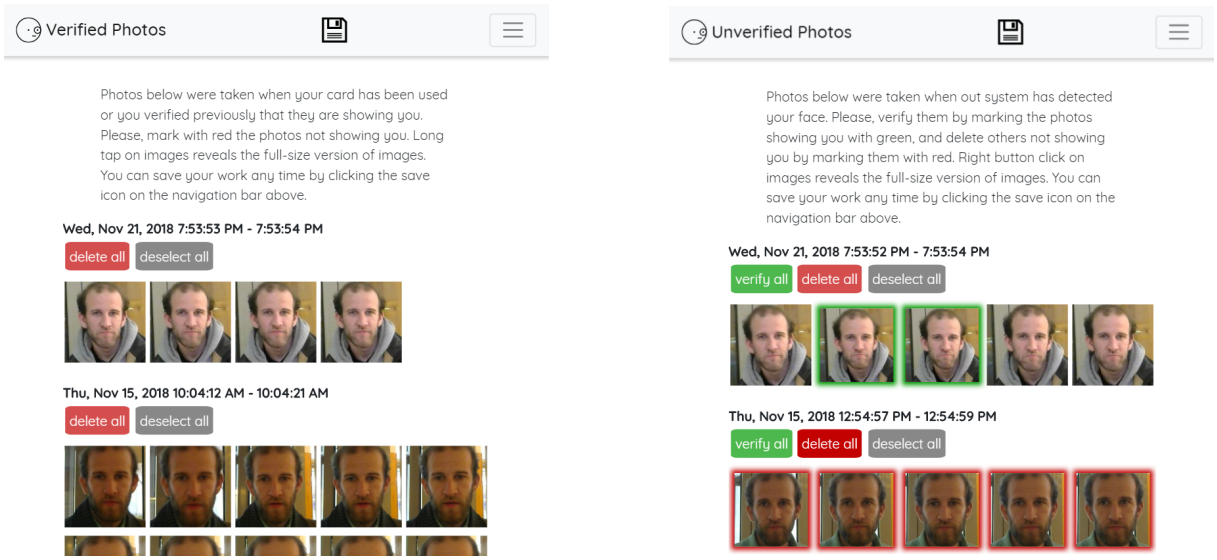


Figure 2.8: Page of verified (left) and unverified (right) photos. Verified images are recorded when the user used their card to get access, and unverified images are recorded when the system detected a face that is similar enough to the user' verified images to trigger gate opening. The main difference between UI of the two pages it that while verified images has only two states (unselected and marked to be rejected), unverified images can be classified into three groups (unselected, marked to be accepted [highlighted with green] and marked to be rejected [highlighted with red]). Change state of the image can be done by click, and user have to click on save button in navigation bar to save annotation.

be rejected, verified images can only be rejected. The term of "accepting" an unverified image refers to the action of re-labeling it as "verified". On the other hand, "rejecting" an image means that it is completely deleted from the database even if it was verified or unverified image in the time of annotation. On the page of verified images, the user can toggle state of individual photos between being unselected or marked to be rejected by clicking on thumbnails. On the page of unverified images, being marked to be rejected appears as a third state. To visualize the current state of the image, a green frame is visible behind images marked to be accepted, and a red one if the image is marked to be rejected. However, the these states are saved only as annotation when user clicks on the saving icon in the navigation bar at the top of the page. To make annotation easier, there are buttons at the top of each group to unselect or mark all photos of the group to be rejected (or accepted). Fig. 2.8 displays screenshots of pages of both verified and unverified images.

Profile page and UI language The website also have a page to display stored personal data such as shibboleth ID, name, email address, and show entries of access log (when did the user enter or leave the building, which gate they used, and what method they are authenticated with (card or face recognition)). That page also has a button dedicated to allow users to delete all of their data stored in our system. Screenshot of that page is depicted on fig. 2.9, and the language switching links are visible on that figure (the page is available in English and Hungarian as well).

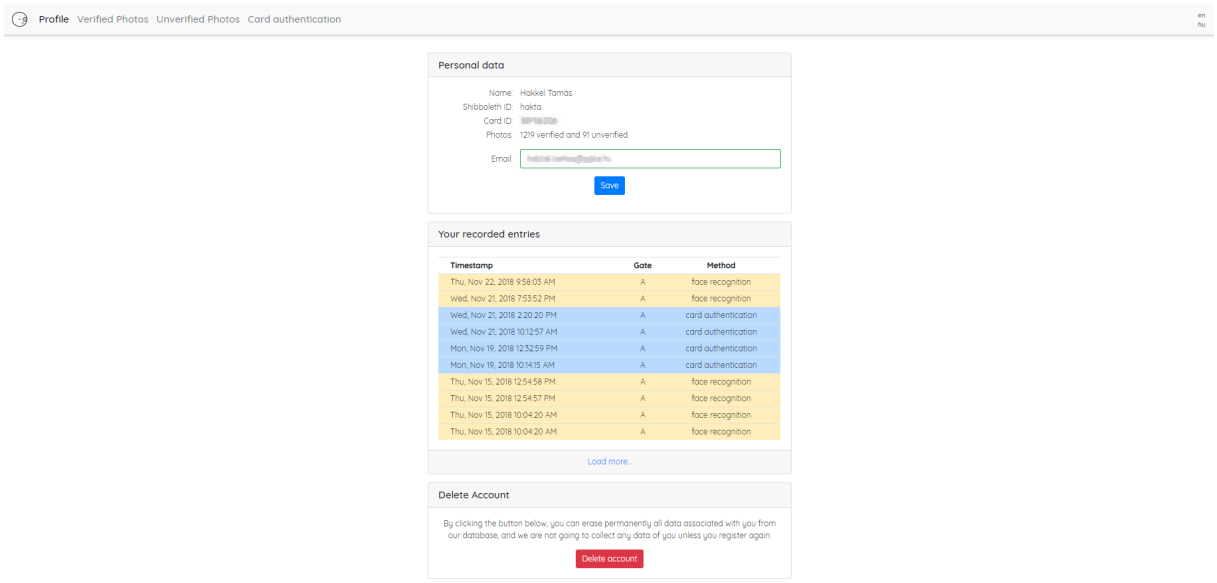


Figure 2.9: Desktop PC view of profile page that displays stored personal data and allows users to delete all of their data stored in our system. In the upper right corner, the language switching links are visible (the page is available in English and Hungarian as well).

Technologies The website consists of static HTML pages that load data with Ajax, and back-end is built using PHP 7.0.30. Front-end design is created with Bootstrap 4.1.0, and JQuery 3.31 is used for DOM modifications and Ajax requests. Source code is available at `https://dev.itk.ppke.hu/sam/website.git`.

2.2.6 Database

Data sources The database of our system is used to collect data from various sources:

- Time, card ID, and card reader name from the Hardware forwarded by Edge,
- Recorded photos with faces and a card ID attached to it from the Cloud, and
- Users' personal data and annotation of photos from the Website.

As we are building a data-intensive application that works real-time, almost all operations needs to store or retrieve data and these operations can be a performance bottleneck. To avoid or decrease that effect, we need to carefully choose Database Management System (DBMS) and design database schema.

Photos One of the most difficult questions we had to answer was that how we should store images. While it is tempting to them as normal files (as we did in the early phase of the development), it turned out later that it is hard to maintain, transfer over network, connect with other data in the database, and restrict access for a given user, so we came to the conclusion

that it is better to store them in the database. Since nearly all modern DBMS supports binary data to be stored as a blob, we had plenty of choices.

DBMS selection Currently, there is two major types of DBMS: classic relational DBMS, and the relatively new noSQL system. While noSQL systems are more flexible and scalable solutions, RDBMS are still widely used because schema normalization is frequently necessary to avoid redundancy, and because of maturity and ACID (Atomicity, Consistency, Isolation, Durability) compliance they are considered more stable and secure [8, 25, 24]. In our case the stability are preferred over scalablity (number of all possible users are still under 1000), and the collected data is strongly interconnected, so we decided to use RDBMS instead of noSQL. The most popular RDBMSs are mySQL, Oracle, PostgreSQL, MS SQL Server, HSQLDB, Firebird, H2, Apache Derby, IBM DB2 and SQLite [15]. The main aspects we had to consider when choosing DBMS were the following:

- The DBMS must be available free of charge.
- A linux version have to be provided.
- PHP and Python wrapper must be implemented.
- Storing large amount of binary data (images) have to be possible.
- Where similar choices are available, speed and simplicity are preferred.

Oracle, MS SQL Server and IBM DB2 are commercial products, HSQLDB, H2, and Apache Derby are mainly for Java applications, so choices are narrowed quickly down to mySQL, PostgreSQL, Firebird and SQLite. PostgreSQL more advanced DBMS than we need, and Firebird is far less documented (and also less popular [31]), and also neither one is installed on the servers of the University. On the other hand, SQLite would be a good choice as it is a lightweight, simple, single-file based DBMS that presents impressive performance under moderate load (e.g. website accesses under 100k/day), and it is available of PPCU servers. However, SQLite is not recommended when the number of concurrent writes is very high, or the size of the database exceeds the maximum file size of the file system [2]. Since we need to store images in database, the size of the database increases rapidly and even after a couple weeks it can reach hundreds of GB of data, so SQLite is not an option for us. MySQL have the advantage of ability to high concurrency and the fact that it is designed to be reached from multiple servers and platforms which is essential for us. It is a wide-spread system, and already installed on PPCU servers, so considering the arguments above, we have chosen to use MySQL.

Schema Our database schema contains 5 tables:

- **user**: That table is used to store data of registered users such as shibboleth ID, name, email, card ID, and chosen language for the Website. To increase speed of Website, we added indexes to shibboleth and card_ID attributes because user selected by shibboleth ID when page loads and card_ID is used to make SQL join between photo, user and card_log table.

- **photo**: Binary data of photos are stored here. We store both the full size image and a thumbnail (thumbnail needed to be pre-calculated and stored in the database to make website load them faster) along with the coordinates of the frame around the detected face (xmin, ymin, xmax, ymax). The field card_ID contains the card_ID of the user associated with the photo. That association can be either by card authentication (photos recorded by the time user swipes their card are connected with them) or face recognition. Finally, attribute is_it_sure is used to distinguish verified and unverified images. Verified images are the ones recorded by card authentication or manually annotated as "accepted" by the user, while unverified images are recorded when face detector recognizes a registered user.
- **card_log**: That table collects data from Hardware: One row is inserted either when user swipe their card or face recognition system triggers gate open. Values of these rows are the following: card_ID, timestamp, gate name, and a 1-bit flag showing whether the gate was opened by card swipe or face recognition.
- **registration_log**: That table is rather just an auxiliary table that records events (actions) of website card authentication, namely page loaded, authentication started, authentication succeded, timed out, and locked out for a day (because of too many authentication attempts). The reason behind that table is that the website needs the timestamp to these events to determine time window and count attempts.
- **annotation**: That table is only present to gather data to prepare statistics. We need this table because rejected images are permanently deleted, and verified images can come from two source (card authentication and manual annotation), so we saved user ID, timestamp of photo taken, timestamp of annotation, source of annotation (card reader or face recognition algorithm), and the annotation itself (accepted or rejected).

Although there are some connection between attributes of tables, we omited to define constraints and foreign keys because these fields are loosely connected (we have to store some data of unregistered users temporarily, and thus there some records without proper connection in other tables (for example in card_log stores card IDs of unknown users to make possible of monitoring access log during card authentication). Also, for performance reasons we wanted to reduce number of constraints and foreign key checks, so we added unique constraints to user table (to attributes shibboleth and card ID), and foreign key to annotation table of attribute card ID. Fig.2.10 shows a detailed diagram of database schema.

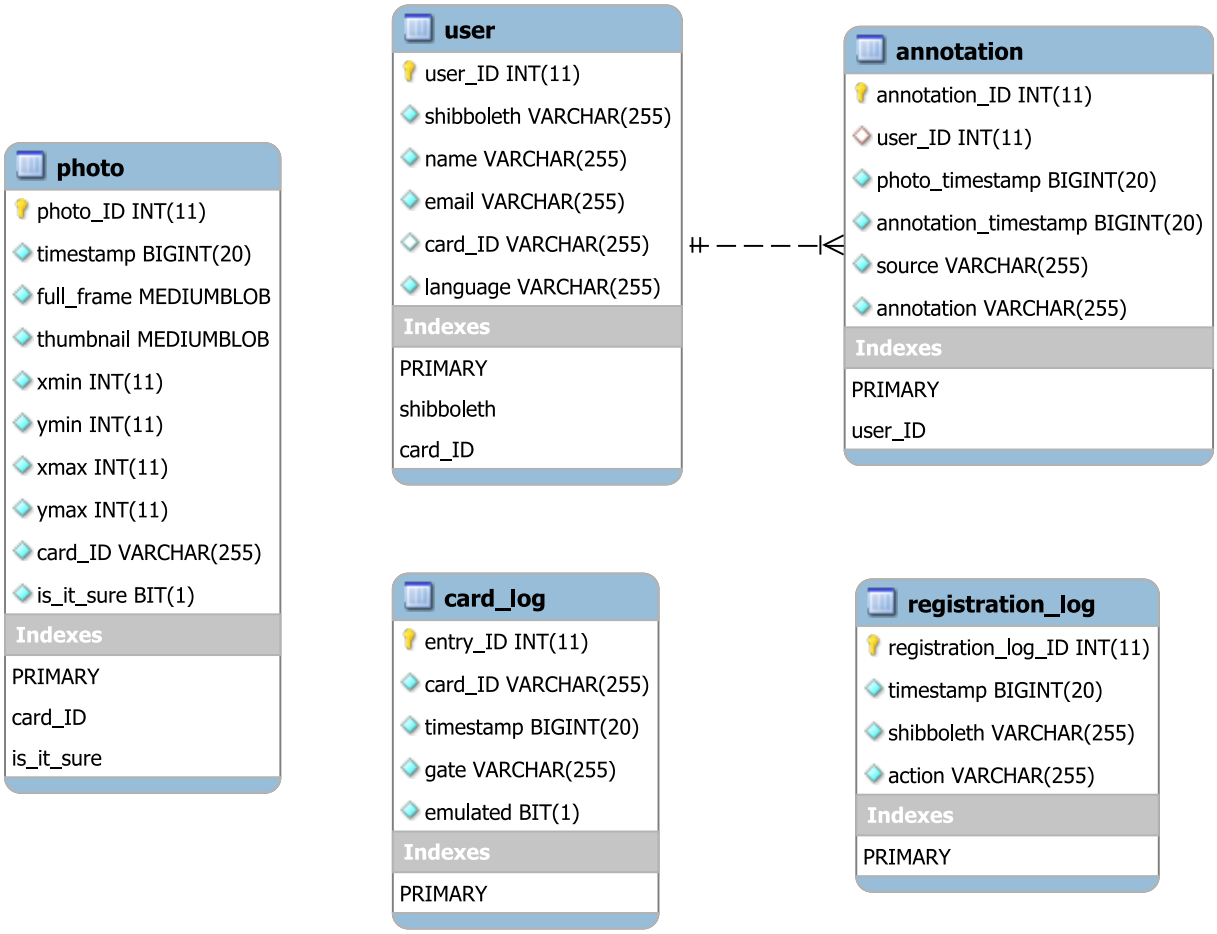


Figure 2.10: Database schema. Exported from MySQL Workbench.

3. Algorithm and evaluation

In our application the face recognition task is solved by a two step authentication: first we identify the user (one-to-many 1:N comparison), next we solve the verification task (one-to-one 1:1 comparison) using subsequent frames. In this chapter we describe the our core algorithm on the following levels: in Section 3.1, we list recent works focusing on large scale face recognition and identification tasks, and report baseline evaluation of their performance on the Sam dataset. In Section 3.2, we detail a robust, non-parametric multi-frame algorithm that treats the single-frame identification algorithm as a black-box predictor. Lastly, in Section 3.3 we describe the dynamically adapting search database that lets our users train the algorithm in real time.

3.1. Single-frame face identification

In this section we provide a detailed description of the identification problem, where a single, unknown *query* face is compared against a *gallery* of known faces, and how is this task is solved using embedding neural networks. The parameter optimization algorithms presented here form the core building blocks of most face-recognition systems and our main contribution in this work is a series of benchmarks with different performance metrics of the cutting edge training methods, implemented and evaluated on our dataset. Firstly in 3.1.1 we list common, open face-recognition datasets that provide real life ("in the wild") images. Second, in 3.1.2 we compare our small dataset to principal training sets in key statistics, such as number of images per subject, resolution, lighting conditions and sample diversity. Lastly, in 3.1.3 we provide a standardized benchmark using the latest state of the art architectures, in different settings w.r.t. the training data: we fine-tune networks which were pre-trained on datasets of increasing relevancy (ImageNET [6], MS-Celeb-1M [9], VGGFace2 [4], respectively).

3.1.1 Dataset review

Increasing sizes Many of the published face-recognition databases and corresponding benchmarks were inspired by the paper, Labeled Faces in the Wild (LFW) [12], which was released in 2007 with $5,749$ subjects and $13,000$ samples. Since then, in parallel with the appearance of deep learning methods in big-data problems (such as the well-known ImageNET [6] challenges), the size of the face recognition databases has grown exponentially. The first open large-scale dataset, CelebFaces Attributes [21], commonly known as CelebA was published in

2014, featuring $10,177$ celebrities on $202,599$ photos. In late 2014, CASIA-WebFace dataset was published in the paper Learning face representation from scratch [36] with $494,414$ images covering $10,575$ identities. Next year, the original VGGFace dataset, described in the paper titled Deep Face Recognition [26] was released, listing $2,622$ subjects on an exceptionally huge number of images, 2.6 million samples. The dataset had been revised and cleaned since its release and it contains 800,000 images with 305 samples per subject according to [4].

Quality over quantity Exponential growth of the datasets has been receding since 2016, due to lagging growth of computation capacity. MegaFace [23] was released in 2016 with $4.7M$ images covering $672,057$ subjects. Even though [23] contains high number of training samples, the image per identity ratio has an average of 7 face per person and reports $> 500k$ distractor faces, which refers to an extremely challenging and mainly unrealistic setting. By the end of 2016, Microsoft published the MS-Celeb-1M (MS1M) dataset [9] that counts $10M$ images for $100k$ subjects. Many papers studying various effects of altering the training conditions [28, 35, 13, 5] report values on [9]. While MS-Celeb-1M is the largest publicly available dataset, its intra-class (or identity) variations is somewhat limited, due to the average 81 samples per subject and because of the relatively high error in the ground truth labels (samples were collected using a search engine without manual correction). To study generalization capability of the algorithms two common *evaluation-only* datasets, IARPA Janus Benchmark-A (IJB-A) [18] and Benchmark-B (IJB-B) [34] were released in 2015 and 2017 respectively. In 2018 VGGFace2 [4] (VF2) was released reporting a modest sized dataset with $3.3M$ images of $9,131$ identities. Even though the size of the training set is limited, benchmarks of [4] show that face recognition networks trained on higher quality samples and labels with less noise outperforms those ones which were trained on [23, 9], not just on the [4] validation set but the IJB-A [18] and IJB-B [34] benchmarks as well. In table 3.1 we summarize the statistics of the described datasets.

3.1.2 Sam Dataset overview

We started data collection by the middle of August of 2018. First, it was tested with a small number of users during summer break, but when the lectures started in September, our system was ready to handle multitude of new users. Unfortunately, from end of September until end of October, we had to suspend the operation of the system for technical reasons, so now there is 44 days (more exactly 332 hours) when we could record data. As of 25th of November, we have the following statistics:

- 171 registered user from both students and teachers.
- $98\,224$ photos of registered users, i. e. $\sim$574 photo is belonging to a user on average. 82.4% of these images are identified by card swipe events, and 17.6% by the face recognition algorithm. On the most busy day, we captured 9019 photos (see Figure 3.1 for more details on distribution of captured images over time). Also, within a day there can be large differences in the number of recorded photos depending on the time of recording (Fig. 3.2 shows that fluctuation).

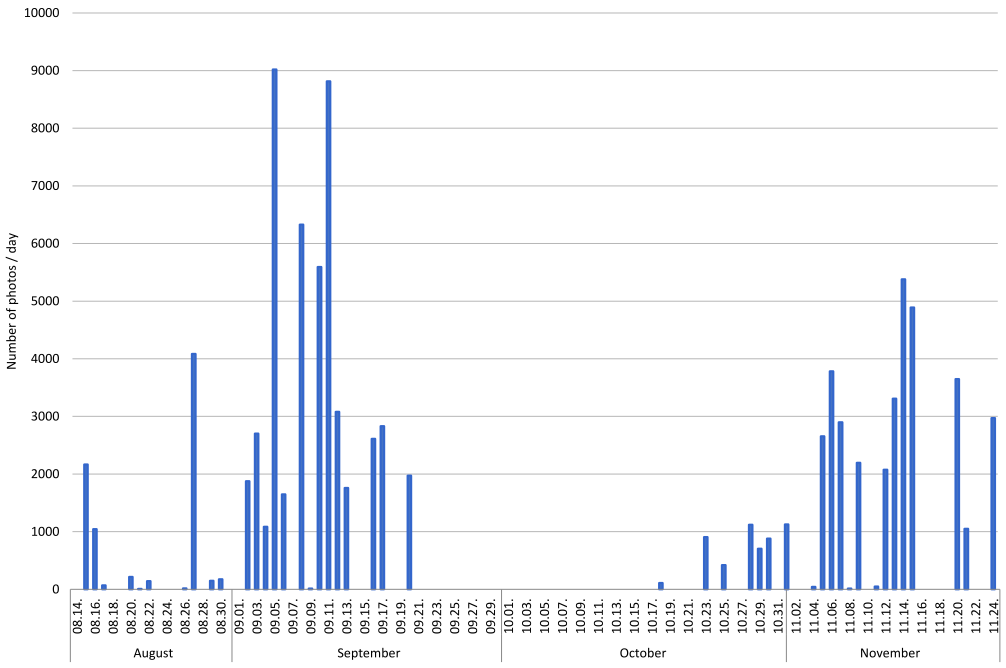


Figure 3.1: Daily number of recorded images over time. We started data collection by the middle of August of 2018. First, it was tested with a small number of users during summer break, but when the lectures started in September, our system was ready to handle multitude of new users. Unfortunately, from end of September until end of October, we had to suspend the operation of the system for technical reasons, so now there is 44 days when we could record data.

- 97.3% (95 601) of these photos are verified photos. Verification could be done either by card swipe event or manual annotation on the website. These verified images are used during face recognition as references.
- Using the Website, manual annotation resulted:
  - $14,617$ images are accepted which were predicted by face recognition,
  - $1,717$ face recognition images were rejected, and
  - $2,280$ photos were rejected which we connected to users based on card authorization events.

The **ratio of accepted and rejected predictions is 89.48%**, which we consider as the overall performance of the algorithm. By the end of September, we began fine-tuning the parameters of the face identification algorithm weekly, so we could manage to reduce number of rejected face recognition photos to 3% of annotated images (see Figure 3.3). On the other hand, rejection of card authentication based predictions are relatively common. That might be due to the incautious habits of users (they use their card to let others in and out the building).

Partitioning the dataset We divide the Sam dataset into four different categories: *training*, *validation*, *test*, *underrepresented*. Firstly, we separate a high diversity set of identities that were covered by more than 50 images each, that sums up to $90k$ samples. We shuffle the high diver-

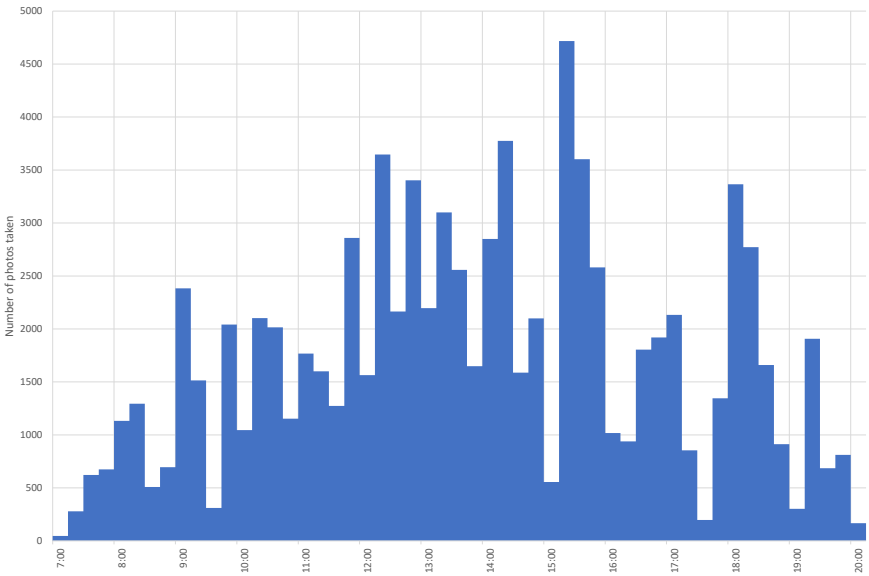


Figure 3.2: Distribution of recorded photos during the day. A column represents the total number of recorded photos during a given 15 minutes long interval.

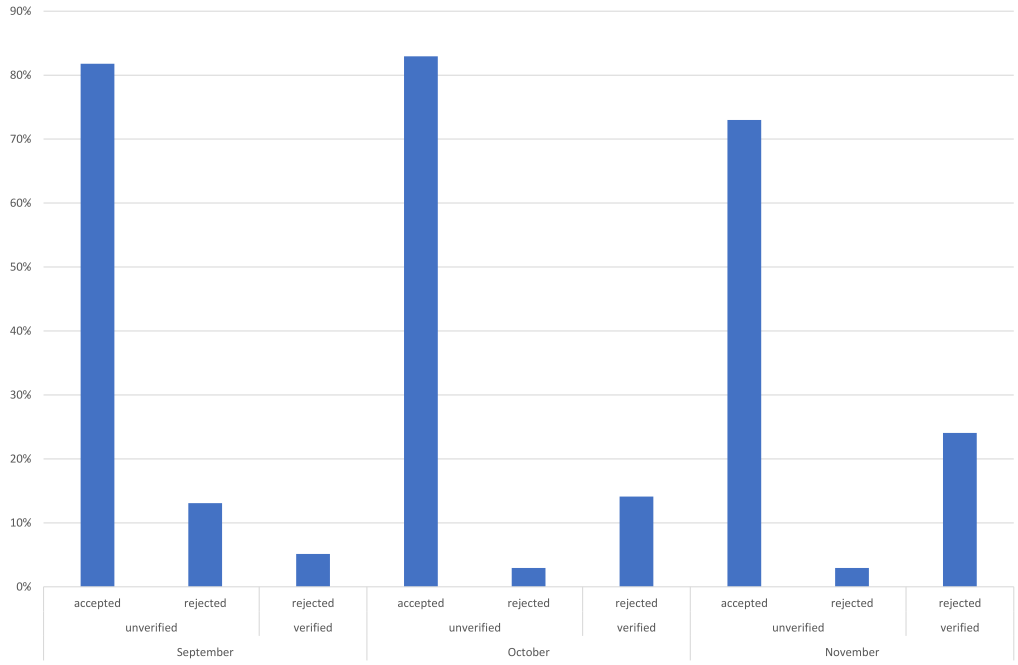


Figure 3.3: Annotation outcomes. Annotation values are grouped by month, so percentage value of each column are relative to the total amount of annotations *in the given month*. By the end of September, we fine tuned the parameters of the face authentication algorithm, so we could manage to reduce number of rejected face recognition photos to 3% of annotated images. On the other hand, rejection of card authentication based predictions are relatively common. That might be due to the insecure habits of users and failures of face tracking algorithm.

Datasets	# of subjects	# of images	# of images per subject	manual	year
LFW [12]	$5,749$	$13,233$	$1/2.3/530$	-	2007
CelebA [21]	$10,177$	$202,599$	-/19.9/-	-	2014
CASIA-WebFace [36]	$10,575$	$494,414$	$2/46.8/804$	-	2014
IJB-A [18]	500	$5,712$	-/11.4/-	-	2015
VGGFace [26]	$2,622$	$2.6M$	$1,000/1,000/1,000$	-	2015
MegaFace [23]	$690,572$	$4.7M$	$3/7/2469$	-	2016
MS-Celeb-1M [9]	$100,000$	$10M$	-/100/-	No	2016
IJB-B [34]	$1,845$	$11,754$	-/36.2/-	-	2017
VGGFace2 [4]	$9,131$	$3.3M$	$80/362.6/843$	Partially	2018
PPCU-Sam (**ours**)	171	$98,224$	$20/574/4048$	Yes	2018

Table 3.1: Relevant entries are read from [4]. Empty values (denoted with dashes) were not reported by the authors. In the '# of images per subject' column min/avg/max values are represented respectively. Partial manual labeling refers to automated clustering where outliers were removed from the data. In contrast, our dataset consists of only manually approved labels, since each sample was accepted by its corresponding owner.

sity set and divide it up in $8:1$ ratio for *trainining* and *validation* purposes (we run a 9-fold cross validation, see the Implementation Details for further information). For the *test* set we select identities, which represent well the scenario when a new user with short history is the subject of identification: the number of samples from the person is > 20 and ≤ 50. We discard identities with images per subject ≤ 20 since they are *underrepresented* and the reported benchmarks would have too high deviance.

3.1.3 Baseline experiments

In our experiments we implement the state of the art algorithm that were used in evaluation of the VGGFace2 [4] dataset. We use common performance metrics to report quantitative analysis on the difficulty of the Sam dataset comparable to others mentioned in 3.1.1. We put a strong emphasis on our objective: we want to increase the performance of the best model on our dataset, to provide a reliable single-frame identification algorithm for our framework, and **not** to improve on the recent state of the art results on either external dataset.

Architecture Similarly to [4], for the backbone feature extractor we implemented the ResNet-50 (later referenced as ResNet) from Deep Residual Networks [10] and SE-ResNet-50 (later referenced as SENet) from Squeeze-and-Excitation Networks [11], winners of the Imagenet Large Scale Visual Recognition Challenge (ILSVRC) [29] 2015 and 2017 respectively. We measured performance under 3 setting for each architecture based on the dataset used for training:

(A) ResNet [10] trained on on MS1M [9] and finetuned on VF2 [4]

(B) SENet [11] trained on MS1M [9] and finetuned on VF2 [4]

(C) ResNet [10] from setting (A) finetuned on Sam

(D) SENet [11] from setting (B) finetuned on Sam

(E) ResNet [10] trained on MS1M [9] and finetuned on Sam

(F) SENet [11] trained on MS1M [9] and finetuned on Sam

Similarity metrics The most straightforward solution for training a classifier is using a feature extractor (in our case ResNet or SENet), a single perceptron layer with softmax activation $\hat{y}$, and training the ensemble with Cross Entropy loss on ground truth labels y:

$$-\sum_i y_i \log \hat{y}_i$$

The first issue arises with the *open-set* classification task when a new, previously unseen class, or in our case an identity is added to the training set: the size of the perceptron layer is fixed and if changed the whole classifier needs to be retrained. In order to avoid this one can remove the classifier completely from the training and use the network to learn a mapping from the input image space to the latent space (the output of the feature extractor) where the distances between the projected samples have meaningful or *semantic* value.

The semantic distance learning is a powerful tool, used for classification without the constraint of the number of classes, rather forming clusters that ideally has low intra-class variance (samples that belong to the same identity are close to each other) and high inter-class variance (samples of different identities are far from each other). In Figure 3.4, we illustrate the learned metric of our model in 2D, using 3 identities and 1000 samples each. Formally, in the space of images $\mathcal{X} \subset \mathbb{R}^{\mathrm{H}\times\mathrm{W}\times\mathrm{C}}$ we denote the learned distance metric $\hat{d} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between sample $x_1, x_2 \in \mathcal{X}$, using an embedding function (our feature extractor) $f_\theta : \mathcal{X} \to \mathcal{H}$, parametrized by θ as the following:

$$\hat{d}_\theta(x_1, x_2) = d\big(f_\theta(x_1), f_\theta(x_2)\big).$$

In the equation above d represents the distance metric that we are going to use in the learned E dimensional latent space $\mathcal{H} \subset \mathbb{R}^{\mathrm{E}}$, formally $d : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$.

There are multiple options for d, the most common choices are the generalized Eucledian-distance L^2 or the inverse cosine similarity $1 - cos(h_1, h_2)$ where $h = f_\theta(x), h \in \mathcal{H}$. The cosine similarity is defined as:

$$cos(h_1, h_2) = \frac{h_1 \cdot h_2}{||h_1|| - ||h_2||}, \forall h : ||h|| > 0.$$

The choice of d determines the training losses we can use for optimizing the feature extractor's parameters θ. For L^2 a common choice is the Triplet Loss, introduced in Facenet [30] which explicitly optimizes the clustering objective, however there are several drawbacks of learning a semantic metric in L^2 space: networks pretrained with cross entropy projects different classes in radial structure (illustrated in Figure 3.5a) and the mapping is unbounded.

Un-normalized embeddings make the optimal threshold selection more difficult, and the train-

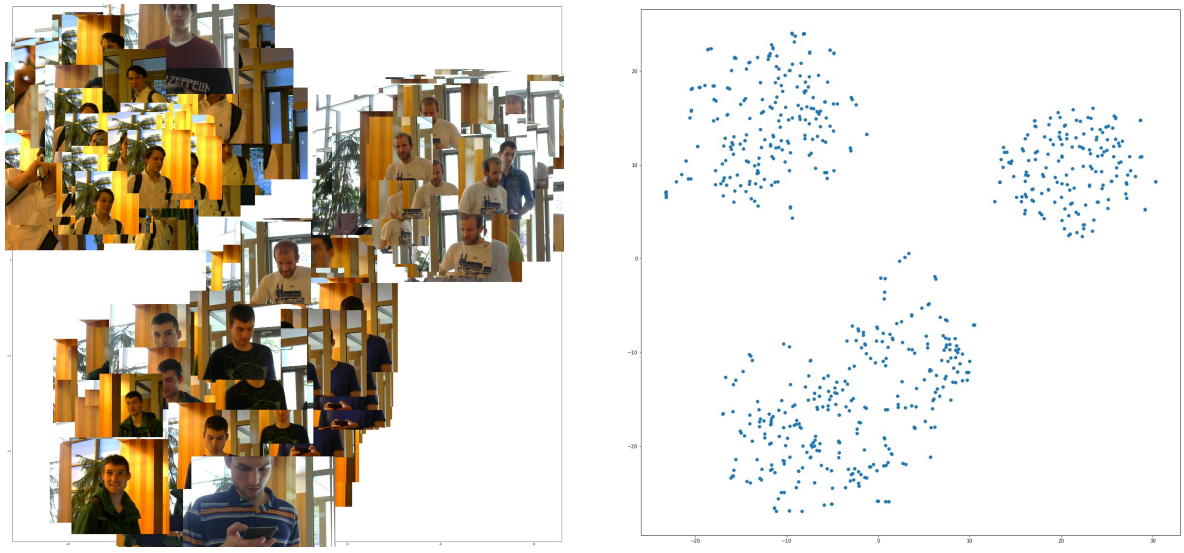


Figure 3.4: We illustrate the learned metric of our model in 2D using t-SNE [22] with 3 identities and 1000 samples each. (left) The samples relocalized to their corresponding projection in the latent space. (right) The latent embedding vectors of each image projected to 2D.

ing requires auxiliary norm regularization techniques. Many recent approach subjects this issue, outperforming vanilla-L^2 metric learning such as: L^2-hypersphere embedding [32], L^2-constrained softmax loss [28] and Center loss [33]. Applying Center loss as an auxiliary training objective along with cross entropy results in a training where clusters are not just distinguished by the cosine similarity, but clearly separable using L^2 metric as well. This effect is illustrated in Figure 3.5b

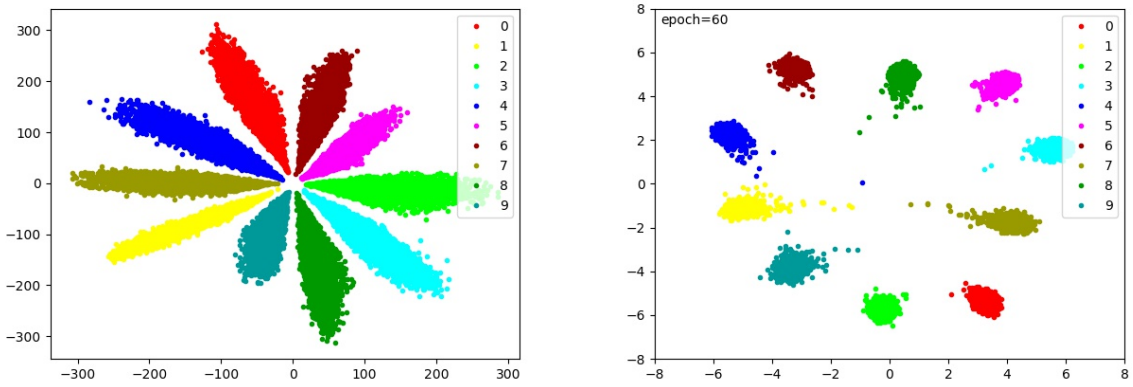
When we are identifying a *query* sample x_q than we compare it against a set of labeled samples (x_i, y_i), namely the *gallery* $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ by the following steps: we evaluate each $d(x_q, x_i)$, then we create a candidate subset of identities $\mathcal{C} \subseteq \mathcal{Y}$, by defining it as "labels of those samples which are closer than threshold distance ϵ", formally $\mathcal{C} = \{y_i | y_i \in \mathcal{Y}, \hat{d}(x_q, x_i) < \epsilon\}$. Finally we count each identical labels' occurrence in $\mathcal{C}$, and define the predicted identity as the most frequent label among the candidate labels.

For its simplicity and for comparable results with [4] we used Cross Entropy loss for training and cosine similarity for all of our experiments.

Performance metrics We use different metrics to illustrate how the architecture and the different pre-training settings influence the outcome, namely:

- fixed False Positive Identification Ratio (**FPIR**)
- Rank-N vs. True Positive Identification Ratio (**TPIR**)
- Accuracy @ k, also known as **Top-k**

Results from the fixed **FPIR** benchmark are shown in Table 3.2. In our case this is the most relevant metric, since it provides us information about the likelihood of identifying correctly the *query* with fixed likelihood of identifying an unknown person as a registered user.



(a) Cross Entropy

(b) Cross Entropy + Center loss

Figure 3.5: Training an MNIST [20] classifier with 2 dimensional latent representation (the input for the last, classifier layer which has an output for the 10 classes). (left) using the classic cross entropy loss function results in radial structure. Notice that how easy it is to find two points that belong to the same class (e.g. a red sample close to the origin and a red sample far from the origin), yet the L^2 distance is larger between them than between two samples of a different class (red sample at the origin is closer to a blue sample at the origin than a red sample at the tip of the cluster). Therefore it is more convenient to use the cosine similarity that, intuitively, tells how well the vector of the two samples align with each other. (right) using the auxiliary Center loss [33] the clusters will behave nicely in an L^2 metric space as well. Credits: https://github.com/jxgu1016/MNIST_center_loss_pytorch

Results from the **Rank-N** benchmark are listed in Table 3.3. To obtain Rank-N we measure the ratio of the retrieved true samples in the N Nearest Neighbour versus the total number of true samples in the gallery. For real time applications this metric is too expensive, because the ranking operation involves sorting the entire *gallery* using the similarity score that has a lower bound on the complexity in the worst case: $n \log n$ for an n size *gallery*. Also, because this metric is sensitive to the high variance of the image per identity ratio, it could be challenging to find an optimal threshold for identification.

Results from the **Top-k** benchmark are listed in Table 3.4. We used Nearest Neighbour search to make predictions for each *query* image. In general, the Top-k result refers to the case when the algorithm makes k guesses and we assume correct prediction if any of the guesses match the ground truth. We note that in our implementation each guess is allowed to belong to any of the identities.

Implementation details For each cross-validation run we use $10k$ samples from the high diversity set for validation and $80k$ sample for training, resulting in $k = 9$-fold cross validation runs in total. By each run (except for setting A and B) we fine-tune the network in 20 epochs on the *training* data and save the state that performs best on the *validation* set, finally we use this state to evaluate scores on *test*. All reported values on the *test* set are computed as the arithmetic mean of the runs.

For each run, we evaluate TPIR and FPIR on the *test* set using the following steps:

1. To get a list of similarity threshold values, we divide the $(0, 1]$ range using 5000 equally sized bins.
2. We compute the maximum number of hits by counting the labels in the *gallery* that matches the *query*.
3. To get the maxmimum number of misses, we take the *gallery* size minus the the maximum number of hits.
4. We select samples that have higher similarity score than the given threshold value to obtain the list of *candidate*s of the algorithm.
5. To obtain the total number of hits we measure how many of the candidate's labels match the *query*'s class.
6. To obtain the total number of misses we subtract the total number of hits from the size of the candidates.
7. To evaluate FPIR at a fixed threshold value we select the total number of hits and divide it with the maximum number of hits. To evaluate FPIR we do similarly.

For optimizing the embedding network we use Cross Entropy loss, Adam [17] optimizer with a fixed learning rate $\alpha = 0.005$, with batch size of 128 on $224 \times 224 \times 3$ images, cropped and aligned with dlib [16]. The output of the embedding network was 2048, for which we applied L^2 normalization in test time. Similarly to [4], we used balancing strategy for each mini-batch due to the unbalanced nature of our *training* data; Monochrome augmentation with probability of 0.2 to reduce over-fitting on colour images. We used pre-trained Caffe [14] weights from the authors [4], publicly available at: `https://github.com/ox-vgg/vgg_face2`. We trained and evaluated our models on 2×RTX-2080 GPU.

Observations In Table 3.2, 3.3 and 3.4 we notice that both the lowest and the highest scores were obtained by using ResNet [10]. The majority of the best results were achieved in setting C in every benchmark, as the result of training the best performing model from [4], using MS1M and VGF and fine-tuning it on Sam-*training*. The worst results were achieved by setting A, B, since they were not allowed to use any of the training samples, validating the usefulness of our limited training data.

To illustrate the small differences our *training* dataset yields, we have plotted TPIR against FPIR on a logarithmic scale in the relevant FPIR range, $(0, 0.1]$ in Figure 3.6a. In Figure 3.6b we compared the best performing setting, C Receiver Operational Characteristic (ROC) curve on our *validation* and on the *test* set. In Figure 3.7, plotting both the TPIR and the FPIR values against the cosine similarity threshold helps us determining the optimal threshold value for test time application. If we want to avoid false alarms (since it causes a serious security breach) we have to trade off retrieval of true matches, resulting in a less successful, or longer identification process. After careful consideration we set the cosine similarity threshold of the single-frame face detection algorithm to 0.6.

Setting	**FPIR**$= 0.01$	**FPIR**$= 0.05$	**FPIR**$= 0.1$
A	0.468 ± 0.351	0.513 ± 0.331	0.571 ± 0.301
B	0.504 ± 0.321	0.546 ± 0.314	$\mathbf{0.578 \pm 0.303}$
C	$\mathbf{0.510 \pm 0.320}$	$\mathbf{0.546 \pm 0.315}$	0.576 ± 0.306
D	0.497 ± 0.319	0.541 ± 0.315	0.572 ± 0.307
E	0.508 ± 0.320	0.546 ± 0.315	0.574 ± 0.306
F	0.504 ± 0.321	0.546 ± 0.314	0.578 ± 0.303

Table 3.2: True Positive Identification Ratios (TPIR) and corresponding standard deviance using fixed False Positive Identification Ratios (FPIR) on SAM-test set.

Setting	Rank-1	Rank-5	Rank-10
A	0.051 ± 0.026	0.150 ± 0.042	0.249 ± 0.234
B	0.057 ± 0.031	0.154 ± 0.063	0.251 ± 0.122
C	$\mathbf{0.060 \pm 0.020}$	$\mathbf{0.163 \pm 0.065}$	$\mathbf{0.264 \pm 0.126}$
D	0.059 ± 0.020	0.160 ± 0.066	0.258 ± 0.127
E	0.059 ± 0.020	0.161 ± 0.066	0.260 ± 0.127
F	0.059 ± 0.021	0.159 ± 0.067	0.257 ± 0.128

Table 3.3: True Positive Identification Ratios using increasing k in the k-NN classification on the SAM-test set. We use cosine similarity for defining the inverse distance between the embedded features. Note the difference between Rank-N and Top-k metrics: using the Rank-N metric we compute TPIR against the rank.

To emphasize the difficulty of the Sam-*test* set we compare the best results of the same algorithm (setting A) in Table 3.5. Notice the high variance of the same algorithm, and the difference in performance, it clearly shows the difficulty of the real world face-recognition task.

3.2. Multi-frame face validation

In this section we introduce our model agnostic, non-parametric multi-frame face validation algorithm, that encapsules the single-frame face identification algorithm, detailed in Section 3.1. The goal is to validate whether the person identified first by the single-frame module can be

Setting	**Top-1 accuracy** $(\%)$	**Top-5 accuracy** $(\%)$	**Top-10 accuracy** $(\%)$
A	95.14	97.28	**98.71**
B	96.15	97.73	98.21
C	**97.48**	**98.36**	98.65
D	96.44	97.90	98.32
E	96.84	98.11	98.53
F	96.15	97.73	98.21

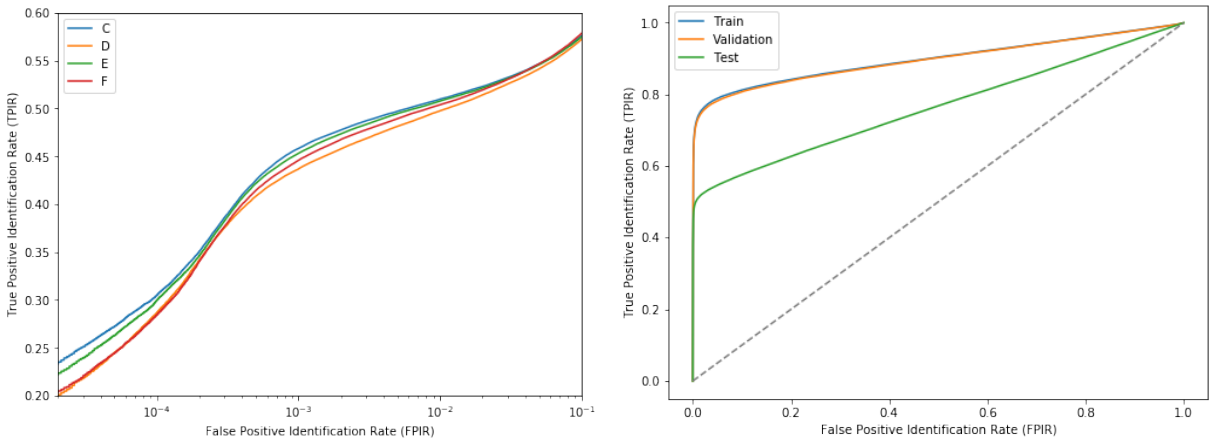
Table 3.4: Top-k performance metric on SAM-test set. Note the difference between Rank-N and Top-k metrics: using the Top-k metric the answer is considered 100% correct if the ground truth label is present in the k first predictions of the algorithm.

Dataset	**FPIR**$= 0.01$	**FPIR**$= 0.05$	**FPIR**$= 0.1$
IJB-B [34]	0.743 ± 0.037	-	0.863 ± 0.032
Sam-*test* (**ours**)	0.468 ± 0.351	0.513 ± 0.331	0.571 ± 0.301

Table 3.5: Baseline evaluation of setting A on the IJB-B [34] dataset (read from [4], Table VII) and our *test* set. Notice the high variance of the same algorithm on our dataset compared to the low variation and the high scores on the IJB-B [34] evaluation dataset. This shows the difficulty of face recognition in real life applications.

Dataset	**FPIR**$= 0.01$	**FPIR**$= 0.05$	**FPIR**$= 0.1$
Sam-*training*	0.736 ± 0.316	0.789 ± 0.300	0.812 ± 0.286
Sam-*validation*	0.729 ± 0.316	0.781 ± 0.302	0.807 ± 0.286
Sam-*test*	0.468 ± 0.351	0.513 ± 0.331	0.571 ± 0.301

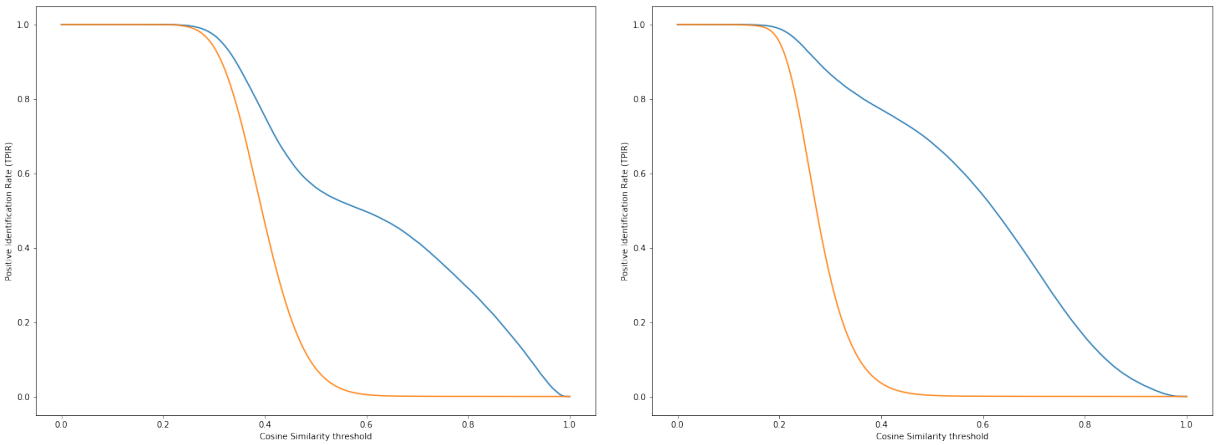
Table 3.6: Comparison of the best performing model (setting C) on the *train*, *validation* and the *test* set. Notice that in real life scenarios results are relevant for identities both in the validation and in the test set, since we the application is not restricted to trained and tested on different identities.



(a) TPIR vs. FPIR on a logarithmic scale.

(b) ROC curve of setting C.

Figure 3.6: Plotting the TPIR values against the FPIR scores yields the Receiver Operating Characteristic (ROC) curve. (left) Plotting different training settings reveals the small gains added by fine-tuning the network on the Sam-*training* set. (right) Depicting the difference between the validation and the test performance. The grey dashed line refers to the random guessing model. For the fixed FPIR comparison, see Table 3.6.



(a) *Test* (b) *Validation*

Figure 3.7: TPIR (blue) and FPIR (orange) values plotted agains the Cosine similarity threshold results in a talkative characteristic: when the threshold $= 0$, every sample in the *gallery* is accepted by the algorithm, therefore both TPIR and FPIR $= 1.0$, while e.g. at threshold $= 0.6$, the ratio of the retrieved negative samples (to the total number of negative samples) is respectably smaller than the ratio of the retrieved true samples (to the total number of true samples). Intuitively the aim is to increase the gap between TPIR and FPIR, by keeping false alarms as low as possible, or push the slope of the FPIR (orange) curve to the left as much as possible.

assigned to a registered identity with high confidence.

Our aim with the presented design was to use the simplest possible approach to utilize the continuous stream of input. The basic idea is to treat the single-frame module as a probabilistic model, and draw multiple samples from the identification process and pass only if successful retrieval was repeated through multiple tests. We use the low temporal variance of the input samples to our advantage, improving the robustness of the final verification process, detailed in Algorithm: 1. We note that the algorithm is vulnerable, for it relies heavily on the identification module's success, however in practice this algorithm is the most straightforward to implement and its accuracy level satisfies our requirements.

3.3. Dynamic gallery

We enhance our system with the ability to extend the face database, or the *gallery* on the fly by a user who wants to train the algorithm, and fine-tune the embedding network f on the extended database later. To adapt to the changes in the search database we divide the tasks in to two categories. The first category is the on-line adaptation, detailed in Subsection 3.3.1, while the second category is the off-line adaptation, detailed in Subsection 3.3.2.

Algorithm 1 Sam Multi-frame verification. The algorithm returns and terminates only when the face is verified and the identity is not a distractor.

Require: $f(x)$ ▷ single frame identification function
Require: $\mathcal{X}$ ▷ stream of frames
Require: N ▷ # of successive retrieval
$y \leftarrow$ <UNK> ▷ initialize identity value
counter $\leftarrow 0$ ▷ initialize counter of successive retrievals
$t \leftarrow 0$ ▷ initialize timestep
while counter < N **do**
    $t \leftarrow t + 1$
    $x \leftarrow \mathcal{X}[t]$
    $\hat{y} \leftarrow f(x)$
    if $\hat{y} \neq$ <UNK> **and** $\hat{y} = y$ **then**
        counter $\leftarrow$ counter + 1
    else
        counter $\leftarrow 0$
    $y \leftarrow \hat{y}$
return y

3.3.1 On-line adaptation

During on-line training, the main goal is to minimize the time between the identification and verification, by maximizing the precision and recall of the single-frame identifier.

In practice, the original samples of the *gallery*, x_i are not stored on the device memory (by device we refer to the the GPU in the *cloud*), only their latent representations h_i. During test time only the embedded code vector of the *query* image is computed. This is detail is crucial from the aspect of real-time application, because its efficiency saves a tremendous amount of compute, and notable portion of the device memory, that leaves room for a dynamically growing *gallery*.

When a user wants to train the algorithm, she needs to stand in the front of the camera and authenticate herself using her unique ID badge. Then our application enters into the training state, which lasts until we lose track of the user using the face detection algorithm (a Convolutional Neural Network, implemented in dlib [16]). In the training state each inferred h is automatically assigned to the identity y, added to the *gallery*. By doing so, the application increases the True Positive Identification Ratio of the identity, since the embeddings of successive frames have significantly higher cosine similarity scores than negative and false positive samples. We note that we do not optimize the parameters θ of the embedding function f_θ during on-line training, and all the freshly added latent vectors h are stored locally on the device.

Using this approach, the time it takes to train the algorithm to identify and validate a new user on a decent level is reduced to 1 minute.

3.3.2 Off-line adaptation

When the face detection algorithm does not detect any face in the input stream for 2 seconds, we begin to synchronize the updated *gallery* with the SQL database, asynchronously with the

face detector, which is allowed to interrupt the synchronization process any time. To increase sample diversity we only store every 3^{rd} image of the user. After the synchronization process, the users can delete from the updated gallery on their personal site, also accept samples that were *verified* by the algorithm. During weekends we fine-tune the embedding network on the latest available training samples in the SQL database. These manual changes take effect once every day (during night hours), by a completely separate process that loads the latest fine-tuned parameters of the network and compute the embedding vectors for every sample in the SQL database.

4. Summary and discussion

To conclude our work in this chapter, we summarize our experiments and how we solved the challenges posed by the real world application. Lastly, we indicate current weaknesses of our solution and suggest directions for future work

4.1. Claims

Our goal was to deploy a real-life automated, face-recognition application that is fully automatized, requiring minimal maintenance efforts. Apart from technical issues, e.g. power outages and connection failures, in our experiments we verify that our original objectives were met. More to that, the Sam framework is not just adapting to everyday usage, but as it collects and annotates data the accuracy of the core algorithm increases day by day. The success of our approach helped popularizing computer vision among our fellow scholars and serves $1/3$ of the estimated regular visitors of the faculty. In 44 days we collected $98,224$ images that allowed us to perform numerous benchmarks to provide quantitative results on our algorithm's accuracy (details in the dataset overview 3.1.2).

4.2. Contributions

Our main contribution, apart from the application, is publishing the source for all modules described in this work. Along with the implementation we are going to release the code used for training the networks using multiple GPUs, the pretrained weights used in our experiments, and the implementation of our benchmarks, to help our peers to review and validate our results. Furthermore, the purpose for making this project open-source is to encourage and attract competitors to challenge our algorithm, to provide better solutions for the application.

4.3. Weaknesses and directions for future work

Multi-frame verification As many pointed out during discussions on the project, the multi-frame verification algorithm (detailed in Section 3.2) is a naïve method, that is depending heavily on the performance of the identification process (Section 3.1). We acknowledge this vulnerability, and we would like to justify our choice: in order to fully understand and maximize

the potential of the framework we needed to break down the experiments to several elmentary steps. This first step, was to measure what are capable of using a parametric model for single-frame identification purposes only.

Multiple server side processes At the time of writing, we are testing the second, completely identical deployment of the application that would face in the opposite direction of our current setting. We designed our framework to be scalable from the aspect of *edge* devices, and the result of deploying a second system would double the growth of our database. However in the current implementation, we run separate processes in the *cloud* for each client, which is inefficient knowing that the dynamic galleries, the embedding networks and the identification methods could be shared between parallel processes.

Bibliography

[1] 2012. URL: https://www.shibboleth.net/index/basic/.

[2] *Appropriate Uses For SQLite*. URL: https://www.sqlite.org/whentouse.html.

[3] Scott Cantor. *Home - Shibboleth Concepts*. Shibboleth Wiki. 2009. URL: https://wiki.shibboleth.net/confluence/display/CONCEPT/Home (visited on 11/22/2018).

[4] Qiong Cao et al. "Vggface2: A dataset for recognising faces across pose and age". In: *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on*. IEEE. 2018, pp. 67–74.

[5] Jen-Hao Rick Chang et al. "One Network to Solve Them All-Solving Linear Inverse Problems using Deep Projection Models." In: *ICCV*. 2017, pp. 5889–5898.

[6] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee. 2009, pp. 248–255.

[7] Eccel Technology. *ib technology - Datasheet*. 2005. URL: https://eccel.co.uk/wp-content/downloads/magswipe_dec.pdf (visited on 11/16/2018).

[8] V Gudivada, Dhana Rao, and Vijay Raghavan. "Renaissance in data management systems: SQL, NoSQL, and NewSQL". In: *Computer* (2014).

[9] Yandong Guo et al. "Ms-celeb-1m: A dataset and benchmark for large-scale face recognition". In: *European Conference on Computer Vision*. Springer. 2016, pp. 87–102.

[10] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[11] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *arXiv preprint arXiv:1709.01507* 7 (2017).

[12] Gary B Huang et al. "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments". In: *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*. 2008.

[13] Rui Huang et al. "Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis". In: *arXiv preprint arXiv:1704.04086* (2017).

[14] Yangqing Jia et al. "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[15] Roland Kender. *Most popular relational databases – 2016 edition – Plumbr*. 2018. URL: https://plumbr.io/blog/io/most-popular-relational-databases-2016-edition.

[16] Davis E King. "Dlib-ml: A machine learning toolkit". In: *Journal of Machine Learning Research* 10.Jul (2009), pp. 1755–1758.

[17] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[18] Brendan F Klare et al. "Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1931–1939.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[21] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[22] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[23] Daniel Miller et al. "Megaface: A million faces for recognition at scale". In: *arXiv preprint arXiv:1505.02108* (2015).

[24] Ameya Nayak, Anil Poriya, and Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases". In: *International Journal of Applied Information Systems* 5.4 (2013), pp. 16–19.

[25] Lior Okman et al. "Security issues in nosql databases". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE. 2011, pp. 541–547.

[26] O. M. Parkhi, A. Vedaldi, and A. Zisserman. "Deep Face Recognition". In: *British Machine Vision Conference*. 2015.

[27] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[28] Rajeev Ranjan, Carlos D Castillo, and Rama Chellappa. "L2-constrained softmax loss for discriminative face verification". In: *arXiv preprint arXiv:1703.09507* (2017).

[29] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[30] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[31] EverSQL Team. *Most popular databases in 2018 according to StackOverflow survey*. Mar. 2018. URL: https://www.eversql.com/most-popular-databases-in-2018-according-to-stackoverflow-survey/.

[32] Feng Wang et al. "Normface: l 2 hypersphere embedding for face verification". In: *Proceedings of the 2017 ACM on Multimedia Conference*. ACM. 2017, pp. 1041–1049.

[33] Yandong Wen et al. "A discriminative feature learning approach for deep face recognition". In: *European Conference on Computer Vision*. Springer. 2016, pp. 499–515.

[34] Cameron Whitelam et al. “IARPA Janus Benchmark-B Face Dataset.” In: *CVPR Workshops*. Vol. 2. 5. 2017, p. 6.

[35] Xiang Wu et al. “A light CNN for deep face representation with noisy labels”. In: *IEEE Transactions on Information Forensics and Security* 13.11 (2018), pp. 2884–2896.

[36] Dong Yi et al. “Learning face representation from scratch”. In: *arXiv preprint arXiv:1411.7923* (2014).

Appendix A. Hardware Schematic