Facial Recognition Biometric System

Project Number: 21-0-203

Developed by Elad Gashri and Yonatan Jenudi
Advisor: Yaniv Julian

Introduction

The Facial Recognition Biometric System is a final project in Tel Aviv University's Faculty of Engineering. It was developed in collaboration with Amdocs.
In this project we developed a biometric system that uses a facial recognition algorithm.
The system was designed to be a real-time attendance system for a large corporation.
The facial recognition algorithm is used to identify the employees as they come to the office and register their attendances.
If a face didn't pass the facial recognition threshold, the live feed GUI allows the employee to identify manually using his ID and employee number.
The system contains 3 types of users: standard employees, admins and a CTO. Each user has different credentials for accessing the system through the REST API.
The system's main functionalities are:

1. Live feed that uses the facial recognition algorithm to identify employees and registers their attendances to the database.
2. REST API that enables the system to be interacted with by an HTTP client to get information about the employees, their images, their attendances and the facial recognition algorithm.

Development

The dataset chosen for this project is "Labeled Faces in the Wild" which includes 13,233 images of 5,749 people. Images of the 2 developers, Elad Gashri and Yonatan Jenudi, were also added to the dataset. The people in the dataset represent the employees of the company. The preprocessing on the dataset was done with Python, Numpy, Pandas and OpenCV. For the augmentation process we used the Albumentations library.
The facial recognition algorithm was developed with Python and PyTorch. The algorithm was embedded in Java using "Deep Java Library" (DJL).
The database was designed as a NoSQL document oriented database and was developed using MongoDB. The database includes 3 collections: employees, images and attendances. Each document in the images and attendances collections includes a reference to a document in the employees collection: employee id.
The Live Feed was developed with Java. The GUI that interacts with the employee in the Live Feed was developed with the Java library Swing. The REST API was developed with Java, Spring Boot and Spring Security.

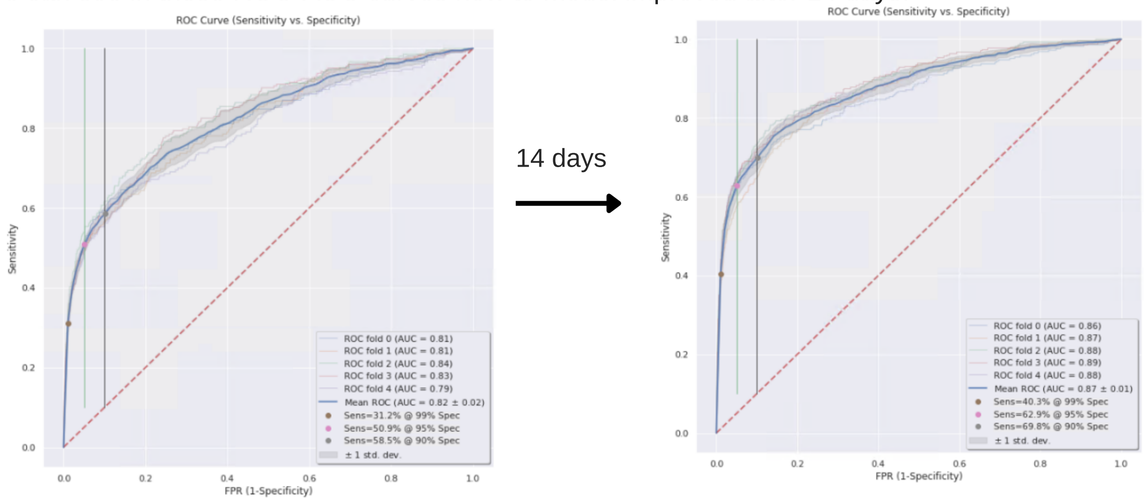
Facial Recognition Algorithm

We used the pre-trained inception Resnet algorithm that was trained on VGGFace2 dataset.
The model is based on the Inception architecture and uses a variety of different layers to optimize its recognition capabilities: Convolutional layers (CNN), Max-pool layers, Dropout layers and fully connected layers.

The model is designed to improve itself over time. We implemented a few algorithms that make sure its learning capabilities will be effective on their own:

1. The model trains from scratch - Every night the model queries all the images from the database. These contain old images of employees who were already recognized by the model and new images of people the model did not recognize (image that didn't pass the recognition threshold). The model trains on these images, and by the morning, the new trained model will be used by the system while keeping a high specificity.
2. Reduce on the plateau -The model knows on its own when to reduce its learning rate and optimize its performance.
3. Early stopping - The model knows when to stop the learning process and to save the state when its performance is at its best.
4. Weighted Random Sampler - The model knows how to handle data imbalance (a problem where one employee has more images than others) and by doing so avoids over-fitting over one employee.
5. Augmentation - When the model learns, it will never see the same image more than once. We invested a lot of time choosing various augmentations techniques to ensure it will know what matters when trying to recognize an employee.

All these features ensure that the system can run independently without having to manually maintain the system.
We can see in these ROC-AUC curves how to model improved after 14 days:



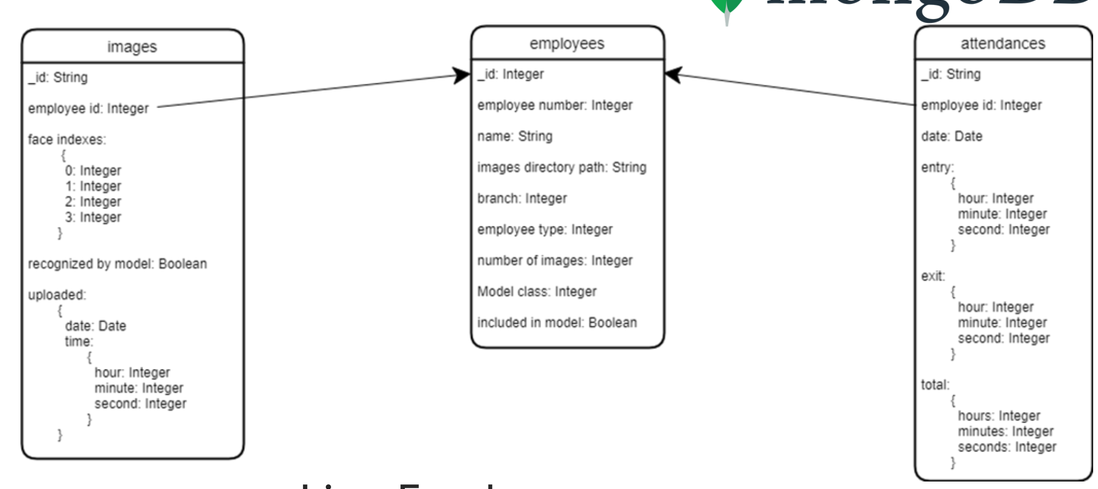
REST API

The API enables performing CRUD operations on the system's database, receiving an attendance report for each employee for a chosen date range and interacting with the facial recognition algorithm.
A standard employee user can only access data about himself by using the API. An admin user can access data about all the employees. The CTO user can also get a prediction from the facial recognition algorithm and train the algorithm.

The API is designed to be consumed by a dedicated UI layer in a future implementation. The API uses a JSON Web Token (JWT) for authentication.
The API was designed with the 3 layer architecture. Each endpoint was implemented with the following design layers:

1. Controller - the presentation layer. It is responsible for correctly routing each HTTP request to the correct Service method and returning the response and an HTTP status code.
2. Service - the business logic layer.
3. Repository - the data access layer. It includes logic for for retrieving, inserting, updating and deleting database data.

The REST API endpoints are:
Login: *POST /login*
Identify employee in image: *POST /model/identification*
Train the model: *POST /model/training*
Get personal details: *GET /personal*
Get personal images: *GET /personal/images*
Get a personal image: *GET /personal/images/{image-id}*
Get branch employees: *GET /branch-employees*
Get attendance report: *GET /attendances?employee-number={employee-number}&start-date={start-date}&end-date={end-date}*
Add or edit entry: *PUT /attendances/entry*
Add or edit exit: *PUT /attendances/exit*
Get employee: *GET /employees/{employee-number}*
Search employee: *GET /employees/search?name={name}*
Verify employee: *GET /employees/verification?id={id}&employee-number={employee-number}*
Add employee: *POST /employees*
Update employee: *PUT /employees/{employee-number}*
Delete employee: *DELETE /employees/{employee-number}*
Get employee images: *GET /employees/{employee-number}/images*
Get employee image: *GET /employees/{employee-number}/images/{image-id}*
Add image: *POST /employees/{employee-number}/images*
Delete image: *DELETE /employees/{employee-number}/images/{image-id}*



Live Feed

The live feed contains to 2 component types: camera stands and facial recognition algorithm stands (A stand means a single processing unit). A camera stand sends recognition requests to an algorithm stand, which returns the recognition results. The communication is based on the UDP protocol. UDP was chosen as the live feed requires low-latency and can tolerate packets losses, because many frames are taken.

The camera stand code was designed with minimal hardware requirements. A camera stand is equipped with a camera, an internet connection and a processor that can perform simple face detections.
The Facial recognition algorithm stand was designed to be multithreaded in order to be able to handle requests from multiple camera stands simultaneously.

When an employee approaches the camera his face will be detected. A CapturedFrame object will be initialized with his frame. CapturedFrame is the class that is consumed by the algorithm. The CapturedFrame object is then sent via UDP to the algorithm stand.

When the UDP packets arrive to the algorithm stand they are enqueued to a multithreaded queue. FramesAnalyzer threads dequeue them. The FramesAnalyzer performs facial recognition using the algorithm, sets the results to the object (face either recognized or not) and sends the CapturedFrame object back to the camera stand it came from. The camera stand displays the result with a GUI that was implemented with the MVC architecture. The camera stand is designed to be connected to a gate controller in a future implementation.

The FramesAnalyzer also enqueues the CapturedFrame object to another multithreaded queue from which an EntryRegister dequeues. The EntryRegister checks if a face has been recognized, and if so, it registers the employee attendance to the database.