SAFELABS

Laboratory User Authentication & User Profile Maintenance System

Mid-Term Progress Report of Data Management Project

CO3554

Data Management Project

2024

Name: Paranawithana T.D.
Index Number: 20/ENG/094
Supervisor: Ms. Dilani Ranaweera

INTRODUCTION

This project proposes the development of a cutting-edge Laboratory User Authentication and User Profile Maintenance System that leverages the power of deep learning to revolutionize how we safeguard and manage authentication to sensitive and hazardous laboratory environments. This system goes beyond normal user-authentication, integrating advanced face recognition, object detection, and user profile management into a seamlessly connected system. Additionally, the records of the resources in the laboratory including safety kits, different kinds of laboratory equipment etc. are maintained by a relational database and the lab occupancy can be viewed as well.

The system employs two cameras strategically positioned at laboratory entry point and the hazardous, sensitive area entry point. As individuals approach the entrance of the laboratory, the system deploys its deep learning-powered face recognition model, instantly identifying authorized users and granting them permission to enter the laboratory. If an unauthorized individual comes, the permission won't be granted. When the permission is granted, it records the attendance of the laboratory user automatically with the timestamp. But security extends beyond mere identity verification as the system needs to ensure the safety of the person entering the hazardous (can be sensitive or may be highly chemical) zone of the laboratory. The user should remove unnecessary accessories that he has with him and wear the necessary safety equipment including the face mask, gloves and surgical hair cap (*Let's say the Personal Protective Equipment – PPE) from the stock of those items kept in a sanitize room before the entrance to the danger zone. Here, at the entry point of the danger zone, a camera captures the individual again. There, the system utilizes object detection algorithms to ensure adherence to critical safety protocols. Whether it's verifying the proper use of masks, gloves, and hair caps (PPE usage), the system acts as a watchful guardian ensuring compliance with established safety regulations before granting full access to the hazardous zone.

Furthermore, the system becomes a central hub for user profile management and laboratory resource management. The system stores comprehensive user profiles including role types and attendance records. Additionally, lab occupancy data

including who is inside the laboratory premises and resource records such as safety equipment availability information become readily available and can be viewed through a mobile/web application, allowing authorities to monitor lab usage patterns and optimize resource allocation.

MILESTONES

1. Literature survey
   A thorough study will be conducted to gather data regarding current technologies and methods. Ongoing studies based on relevant subjects will be considered.

2. Data collection and preparation
   Datasets of images for the face recognition model and safety equipment detection model will be collected and data preprocessing will be done.

3. Select, develop and train face recognition model.
   Most suitable deep learning model should be selected and developed according to the system architecture requirements. Also, training the model should be done with a proper set of data.

4. Develop and train object detection model to recognize face mask, surgical hair cap and other accessories. (PPE kit detection)
   An object detection model should be developed to recognize face mask, surgical hair cap and other accessories.

5. Mobile/web Application Development
   A user-friendly mobile/web application will be developed for lab occupancy viewing, resource management and equipment availability checking for the authorized lab users.

6. Database Integration
   A database will be put into place to securely handle and preserve the information of user profiles, attendance, and resource records.

7. Testing and Evaluation
   Unit testing and system testing will be done thoroughly with several test cases in order to implement a more accurate and effective system.

WORK PROGRESS

Literature review was done about the current technologies and methods used in user authentication in environments like laboratories. Also, research was done about the face recognition models which are pre-trained and available online.

The drawbacks of Manual Processes:

- Reliance on physical access cards: Lost or stolen cards can compromise security, while forgotten cards can hinder authorized access.
- Paper-based attendance records: Prone to human error and time-consuming to maintain, leading to inaccurate data and potential safety concerns.
- Visual inspections for safety compliance: Subjective and susceptible to oversight, potentially leaving room for safety breaches.
- Limited scalability: Manual processes struggle to adapt to larger facilities or increased personnel, leading to logistical challenges and compromised security.

As the complexity of laboratory work grows, so does the urgency for a more robust and automated solution. The consequences of inadequate authentication control and lack of strong compliance can be dangerous, ranging from accidents, injuries and environmental damage. Furthermore, the sheer volume of data generated in modern labs needs efficient management and analysis. Manual methods simply cannot keep pace with the demands of real-time monitoring, resource allocation and incident response.

Face recognition-based user authentication in this project consists of two parts or attempts.

1. Building a face recognition model from the scratch using Convolutional Neural Network (CNN) and train on a fixed number of users
2. Using a pre-trained face encoding generator model which uses deep learning and use it for the image data.

The initial idea was to build a new face recognition model from scratch using Convolutional Neural Network (CNN).

Here, before face recognition, face detection was considered first. For face detection, I found two methods.

I. **Haar Cascade**: The Haar Cascade classifier is based on Haar-like features, and it works by detecting features in images through a cascade of classifiers trained to recognize common patterns in faces. It's fast and suitable for real-time applications but tends to struggle with variations in lighting, angles, and smaller faces in images.

II. **MTCNN (Multi-task Cascaded Convolutional Neural Network)**: MTCNN is a deep learning-based approach that combines three neural networks for accurate face detection, even in challenging conditions such as varying poses and lighting. MTCNN detects faces and landmarks by progressively refining regions likely to contain faces, making it more robust for detecting faces with varying orientations and expressions.

Table 01: Comparison between face detection methods

Feature	Haar Cascade	MTCNN
Algorithm Type	Traditional, feature-based	Deep learning-based
Detection Speed	Faster	Slower but optimized
Accuracy	Lower accuracy in varied lighting and angles	High accuracy with varied lighting and angles
Handling Small Faces	Limited	Effective, due to multi-stage detection
Robustness	Limited robustness for complex scenarios	High robustness, supports face alignment
Ease of Implementation	Simple, requires minimal setup	More complex
Use Cases	Quick, basic face detection in controlled settings	Applications requiring accurate detection under real-world conditions

Initially, Haar Cascade Classifier was used in the attempts in this project due to its simplicity, ease of use, and speed. However, MTCNN was chosen to do face detection after the first few test attempts.

Data collection and preparation

A dataset for detecting safety regulations which is detecting personal protecting equipment (PPE) was found in https://huggingface.co website and it contains nearly 12,000 images.

For the face recognition model, there are a set of datasets in online platforms like https://www.kaggle.com including Labeled Faces in the Wild (LFW) and Celebrity Faces Dataset etc. But for the moment, it was not suitable for this scenario since a new CNN model training was required and those datasets contain a smaller number of images per a class and may be erroneous for a real-world scenario where several people being used. (Famous CNN models including ResNet-34, ResNet-50 and InceptionV3 and face recognition models like FaceNet has been pre-trained with more than one million images.)

Therefore, a custom image dataset was created for the training and testing purposes of the CNN models built in this project with several classes that are several people names including faces of those people. Here, I used 3 or 4 classes for training and testing different models which includes myself, my several friends etc. And used the images of Hon. President of Sri Lanka as it was easier to find the video clips and extract images from them.

First, I used webcam to capture images and tried with 3 classes with around 100 training images and around 500 testing images per a class. The images were stored in 3 class directories which were inside rain and test directories. Images were given unique IDs using a python code and face cropping was done using Haar Cascade Classifier. (This was used in the first attempt but in next steps, MTCNN was used to crop the faces.)

Image visualization and preprocessing was done, and images were normalized so that the pixel values ranging 0 to 255 became in between 0 and 1. Image size was fixed to be 224 by 224. And the images were converted to tensors.

Develop and train face recognition model.

Tensorflow deep learning framework was used to build and develop the models.

A basic model was built initially with four convolutional 2D layers having filter size 10, kernel size 3 and ReLU activation function (Since Rectified Linear Unit (ReLU) function makes the negative inputs 0 and keeps the positive inputs as they are.) and two max pooling layers to reduce the dimensionality and extract the most important features from the previous layer and also a Flatten layer for the hidden layers. The output layer contained 3 neurons for the 3 output classes (‘Althaf ’, ‘Dushan” and ‘Thithira’ are the class names) and Softmax activation function.

The model was compiled with Adam optimization and the learning rate was set to the default.

The model was trained for just five epochs first with the image dataset and then did improvements. Model got training accuracy above 94% and validation accuracy around 98% but both accuracy and loss seemed to have different ups and downs and no smoothness or sharpening was shown. Therefore, the loss and accuracy curves looked worse, and model didn’t do well in custom images.

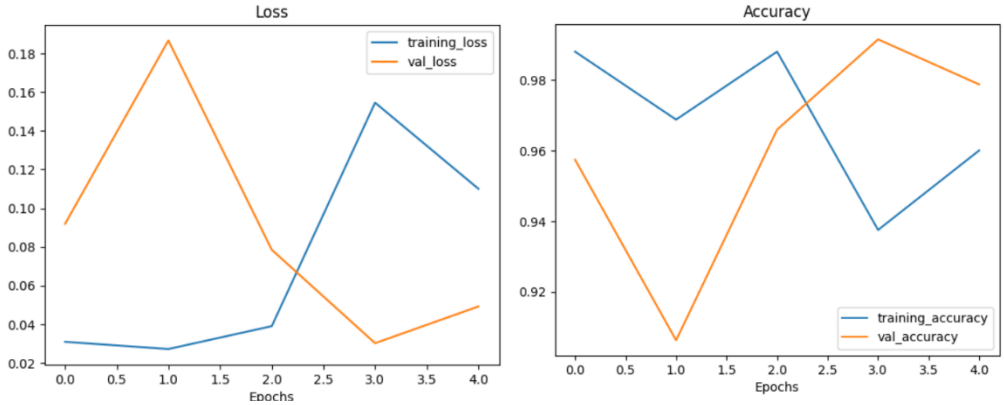

Figure 01: Loss & Accuracy Curves of the initial model

This displayed that the model has not learned well, and the model has memorized rather than learning general features, resulting in near-perfect accuracy but poor generalization. Overlapping between the training and testing data, smallness of dataset and non-diversity of data may have caused this problem.

To address this issue, training set was increased to 500 images per every class and the images were converted to grayscale for simplicity before feeding to the model. This made the model to display both training and validation accuracies to depict about 100% from the second epoch and continued to be so. Training loss instantly became very low while the validation accuracy increased. This model also did not perform well on custom image data testing and failed to generalize. So, the same issue continued as model has memorized rather than learning general features.

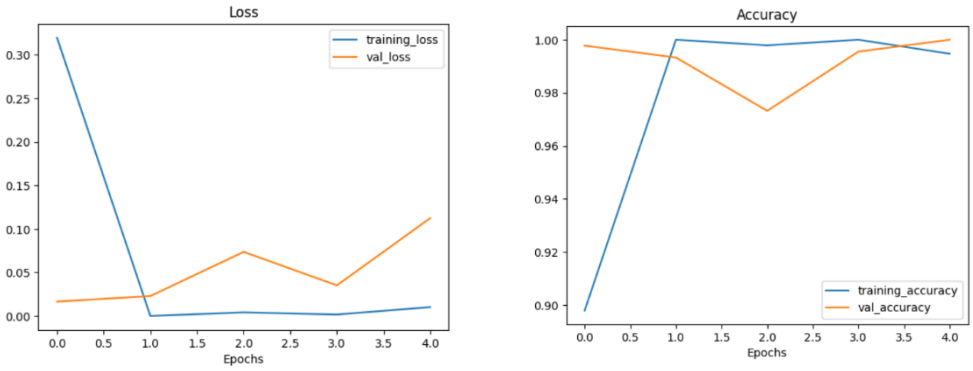

Figure 02: Loss & Accuracy Curves of the above model

Following adjustments and changes were done during data collecting, data preprocessing, model buildings and fitting to the model.

- Replacing the images of the dataset was done as the images captured using the webcam did not have a higher quality. It was needed to feed high quality images for training the models. Therefore, video clips of several people were captured using mobile phone and the images were extracted from those video clips.
- Used RGB colored images.
- Shifted to the MTCNN face detection method which gives higher accuracy instead of using Haar Cascade Classifier.
- Used data augmentation to have more diversity to the image data. Here, rotating, zooming, shifting, and flipping etc. were done to augment the images.
- Also, model was trained for more epochs and checked the accuracy values in several times. (It was trained for 20 epochs with early stopping callback and then for 70 epochs to check how it works.)

With these changes, still the model showed around 100% for both training and validation accuracy. The loss and accuracy curves seemed to have improved a little but without smoothness.

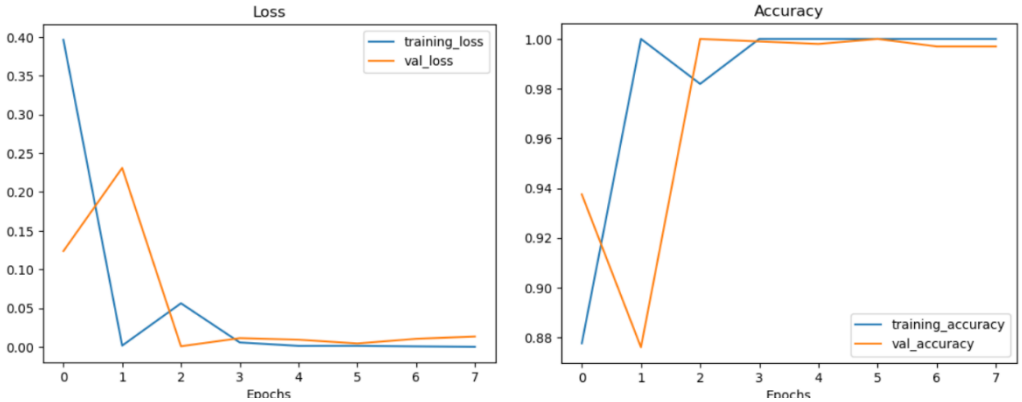

Figure 03: Loss & Accuracy Curves after data augmentation

Also, it did perform a bit better on custom testing images than the previous.

And now the training set was increased to 700 images per class for the next attempts. Test set was reduced to 300 images per a class.

Here, **Transfer Learning** was used since it seemed like the dataset was not enough to train the built models. Since that, pre-trained deep learning models were tried in order to leverage their existing architectures as they are already trained on image data. ResNet and Inception are the models considered in FaceNet model and so as in this project.

ResNet50V2 model was the first one to try. It is a deep learning CNN model which was trained on over 1 million images from ImageNet dataset. ImageNet dataset is a large-scale dataset widely used for training and evaluating image classification models. It contains over 14 million labeled images across 1,000 classes.

Feature extraction was done from the ResNet50V2 model without training the layers of it from the dataset I created. And then the output of that base model was sent to a Global Average pooling layer which reduces the dimensionality to create a feature vector and connected the output Dense layer.

Adam optimizer was used again as the optimizer. And fit the model for 10 epochs, 20 epochs and the previous issue was occurred again as model seemed memorized rather

than learning since 100% accuracy achieved soon without doing well on custom image testing.

Again, data augmentation was done to address the issue. And this time brightness level range also adjusted in augmentation. Dropout layer was added with 0.5 probability for regularization to reduce overfitting if overfitting is the issue and trained for 30 epochs. This time, the feature extraction model seemed learning something as the training and validation loss gradually went down from 1.44 to 0.94 and 1.35 to 0.76 respectively. But still model could not recognize the custom test images correctly with a higher probability of confidence.

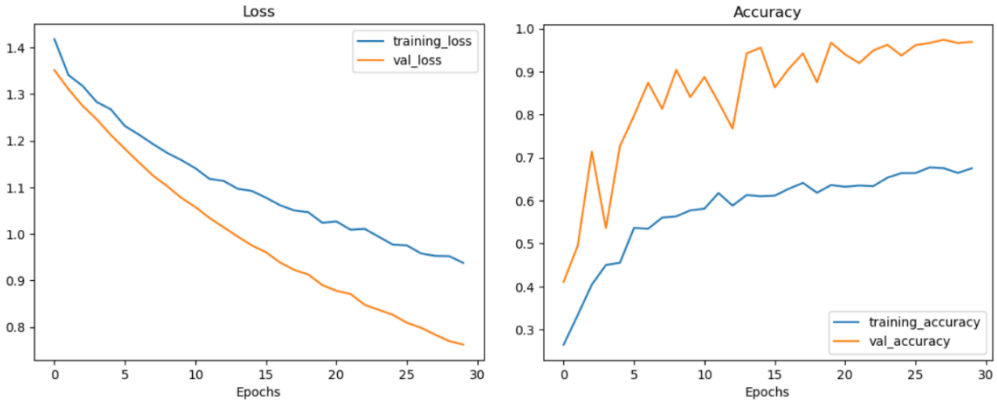

Figure 04: Loss & Accuracy Curves of the above model

These kinds of several changes were done to the model and data and checked how the accuracy and loss goes. (Here, augmentation parameter tuning, training for more epochs, layer changes including dropout level change were done.) But not much improvement was seen.

Therefore, fine tuning the ResNet50V2 model was the next step. Here, the last few layers (8 – 10 layers in several attempts) were unfrozen (trainable) to be trained on the dataset I created and kept all the other inner layers frozen. Leaning rate was set to be 10x lower than before for fine-tuning as model is updating already learned weights. Even with fine-tuning the ResNet model, no better improvement could be seen.

InceptionV3 was the next model to test with. InceptionV3 was also pre-trained on over 1 million ImageNet images. It contains more layers than ResNet50V2.

Feature extraction was done from the InceptionV3 model without training the layers of it from the dataset I created. And then the output of that base model was sent to a

Global Average pooling layer which reduces the dimensionality to create a feature vector and connected the output Dense layer. Three output neurons were used here for the three output classes. The whole model was fit with the dataset I created for 20 epochs and again model seemed to be memorized rather than learning the features.

Again, data augmentation was tried to improve the model and the same model was fit with augmented images for 20 epochs. The result became much better as the loss and accuracy curves looked to be the best ones so far with the smoothness. It seemed that the training and validation results were becoming much closer. It achieved about 97% validation accuracy on test data with training accuracy being slightly lower. And the validation loss was around 0.3 while the training loss is slightly higher than that. This model did perform well on custom images as well with recognizing 73.7% faces accurately from 19 custom images.

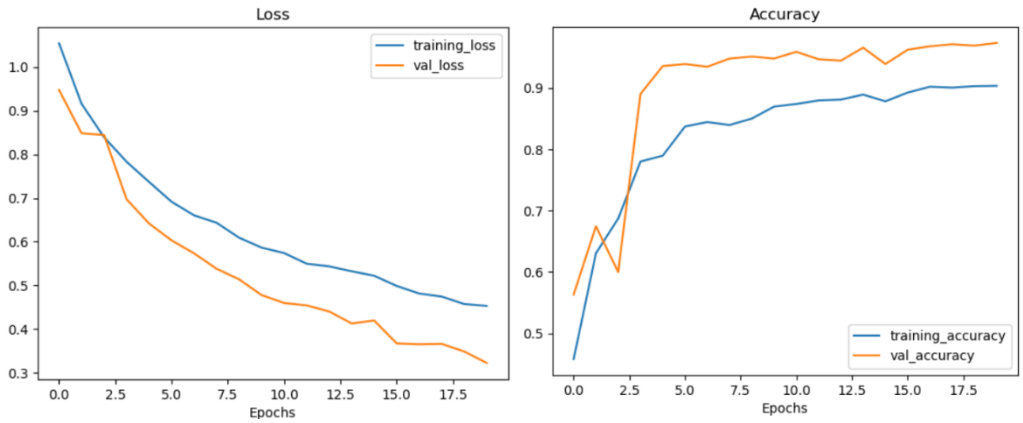

Figure 05: Loss & Accuracy Curves of the last model

So far with these results, it is understood that creating a near-perfect face recognition model from scratch which performs best on custom test data without errors, is not easy. It even takes a lot of time to achieve a higher percentage of accuracy and need of doing a lot of experiments. So, for the moment it looks that this does not suit for going forward with the project. And another concern here is, when a new user registers to the laboratory, it is needed to re-train the face recognition model with including the new user's image data. This may affect the accuracy of the model badly and it is not much efficient and easy to go with this. So, the solution is,

Using a pre-trained face encoding (embeddings) generator model.

Using a pre-trained deep learning model which generates face encodings of the input faces solves the issue of re-training the model with image data. The image data belongs to the different classes should be provided to the face encoding generator model and the model, which was pre-trained, generates the face encodings (embeddings / a numerical representation of the image) for the input images. When the camera detects a face, the model generates the embeddings of that face image and compares that with the embeddings generated and saved earlier and then verifies whether there is a match. This even works well with a single face image per a class, having around five images per a class will make it more accurate. And when a new user registers, it will need only to capture and save a few images of the user with a unique image ID and generate the face encodings of those images. Then Euclidean distance method can be used to compare the face distances.

Below is a comparison of characteristics between famous deep learning face encoding generator models which are used for face recognition.

Table 02: Comparison between face recognition models

Feature	face_recognition (dlib)	FaceNet	VGGFace	ArcFace
Accuracy (on LFW)	~97-99%	~99.63%	~97-98%	~99.83%
Speed	Fast with HOG Moderate with CNN	Moderate	Moderate	Slower
Architecture	ResNet-34	Inception - ResNet	VGG-16 / VGG-19	ResNet-50 / ResNet-100
Model Size	Small to Moderate	Moderate	Large	Large
Use Cases	Face recognition and verification, embedded applications	Face verification, clustering, embeddings generation	Face verification and classification	Face verification, high-accuracy face matching
Embedding Dimension	128	512	4096	512
Ease of Integration	High, easy with Python API	Moderate	Moderate to complex	Complex
Scalability	High, works with relational databases	Medium (best with specialized vector DBs)	Low	Low
Library Dependency	face_recognition (dlib)	TensorFlow, Keras	Keras, PyTorch	PyTorch, MxNet

All the four deep learning-based face recognition models compared here shows a good accuracy. FaceNet and ArcFace offer slightly higher accuracy on specific datasets but often require more processing power and may need retraining or a more complex setup for large-scale use.

VGGFace has higher embedding dimensions, which requires more storage and processing for similarity comparisons, and often is slower in real-time applications.

In this project, where adding users without retraining and maintaining good accuracy is key, dlib's face_recognition offers a balanced solution for speed, accuracy, and operational ease. Therefore, dlib's face_recognition is used for face recognition in this project.

face_recognition library

The face_recognition library, built on top of the Dlib library, uses deep learning to locate and recognize faces in images and videos. The library primarily relies on two key operations: face detection **and** face recognition**.**

1. Face Detection:

    - For detecting faces, the library uses HOG (Histogram of Oriented Gradients) or a CNN (Convolutional Neural Network) model, both available in Dlib.
    - HOG is the faster option and suitable for real-time applications but may be less accurate in complex scenarios. It is an effective method that extracts features from images by analyzing gradient orientation and intensity changes.
    - The CNN model is relatively more accurate but slower and typically used for applications where accuracy is more critical than speed. It requires more processing power as well.

2. Face Recognition:

    - After detecting faces, the face_recognition library uses Dlib's deep learning based ResNet-34 (Residual Neural Network) model, trained on a large dataset, to extract facial embeddings. Embeddings are a mathematical representation of the face that the model can compare for similarity.

- Each face is transformed into a 128-dimensional vector, capturing unique facial features. This vector can then be matched against other stored vectors (from known faces) for identification.

Advantages of face_recognition library

- Easy to Use: face_recognition offers a high-level API that simplifies face detection and recognition in just a few lines of code.
- Good Performance: It offers quite good accuracy and performs well for many real-world applications.
- No Need for Extensive Training: The library can identify faces without additional training, as it leverages pre-trained models, making it ideal for rapid deployment.
- As normalization is handled by the library internally itself, rescaling is not needed and unless the image size is exceptionally large, it is not needed to downsize them.

Implementation of face recognition

For the moment, the image files were stored locally in a folder called ‘Images’ and those were given image IDs. 3 images each from 3 classes were saved to achieve higher accuracy. And the face_recognition library was used to detect faces and generate facial encodings. Here, HOG method was used for face detection as it is faster to detect faces. A Python script was executed using PyCharm IDE to detect faces, generate face embeddings (encodings) for each image in the folder and save those facial encodings to a pickle file.

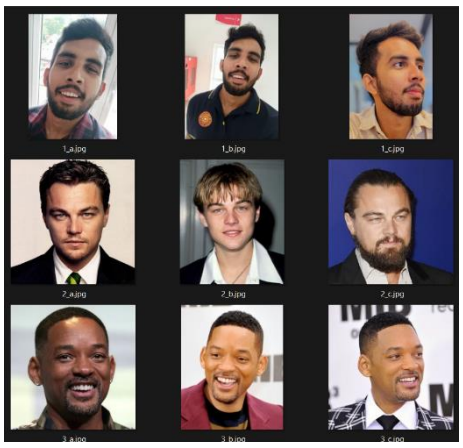

Figure 06: Example of saved images

Then, OpenCV was used to capture the live video of the webcam and again using face_recognition library, face detection from the frames and generating face embeddings for them was done. Then face embedding distances of the current video frame were compared with the face embeddings saved in the pickle file using Euclidean distance method to find whether there is a match. (A machine learning model like SVM can be used for classifying these embeddings but that may require re-training it when a new image is saved. Also, it assumes that the number of users is not a fixed amount and new users may register.)

A threshold value of 0.55 was added and only if the lowest face distance value was below the threshold, the system confirms a matching in order to avoid an unknown face being recognized as a bit similar known face.

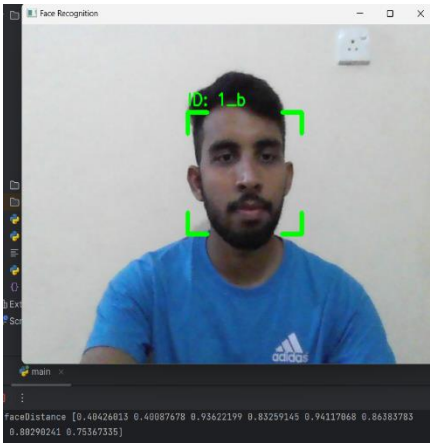

Figure 07: Testing on a known face

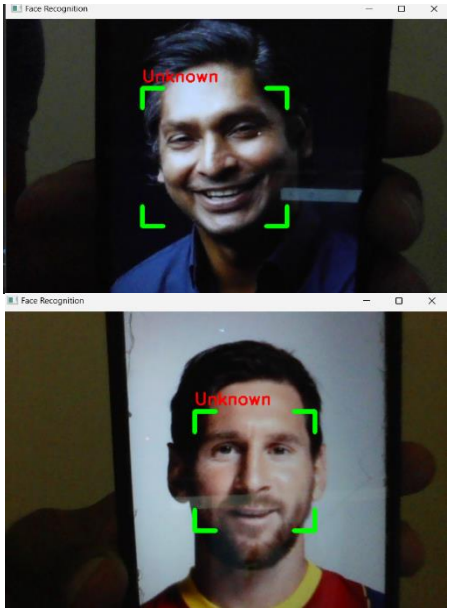

Figure 08: Testing on unknown faces

When a new user registers, the system does not require re-training anything. Only we have to do is, save clear face showing images of him/her and run the face encoding generator to generate the facial encodings for the newly saved images. Here, the encodings are generated only for the newly added images and updated the pickle file to improve the efficiency.

Database design and integration

MSSQL which is a relational database is used to save attendance records of the lab users, user details, user roles, resource details including safety equipment, user image IDs, announcements etc.

Below are some reasons to choose MSSQL as the backend database for this project.

1. Advanced Features: MSSQL offers advanced security and built-in support for row-level security, making it suitable for sensitive data like attendance records and user details.
2. Integration Capabilities: MSSQL integrates seamlessly with other Microsoft tools like Visual Studio and .NET technologies. As C# with .NET CORE framework and Visual Studio will be used for backend development of the mobile application, MSSQL is an ideal fit.
3. Efficient Query Performance: MSSQL optimizes query execution plans and has better scalability for handling large datasets, such as logs of attendance and resource details.
4. Ease of Maintenance: With tools like SQL Server Management Studio (SSMS), MSSQL provides an intuitive interface for managing and querying databases, ideal for ongoing project updates and management.
5. Reliability and Support: MSSQL is widely used in enterprise applications, ensuring access to comprehensive documentation and support.

Here are the details of how the database is structured:

1. User Table
    - Fields: Id (Primary Key), role Id, name, email, password, NIC, address, contact number etc.
    - Purpose: To store information about lab users, including their roles and credentials.
2. Role Table
    - Fields: Id (Primary Key), role name (e.g., Admin, Normal user).
    - Purpose: To define user roles and access levels within the system.
3. Attendance Table

- Fields: Id (Primary Key), user Id (Foreign Key), date, check-in-time, check-out-time.
- Purpose: To track user attendance when they enter and leave the laboratory.

4. Resource Table
    - Fields: Id (Primary Key), resource type Id, brand, price.
    - Purpose: To manage and monitor the laboratory resources including availability of safety equipment.
5. Resource Type Table
    - Fields: Id (Primary Key), resource type name etc.
    - Purpose: To define laboratory resources within the system.
6. Image Id Table
    - Fields: Id (Primary Key), user Id (Foreign Key), Image Id
    - Purpose: To store image Ids for authentication.
7. Announcement Table
    - Fields: Id (Primary Key), description etc.
    - Purpose: To communicate important information to lab users.

Key relationships:

- Id (user Id) in the User Table serves as a foreign key in the Attendance Table and Image Id Table.
- Id (role Id) in the Role Table links to the User Table, defining user permissions.
- Id (resource type Id) in the Resource Type Table serves as a foreign key in the Resource Table.
- When a user comes near the camera and a match of face encodings to the saved encodings is confirmed by the program, the attendance with the timestamp is stored as a record in the attendance table in the database with the aid of the matched Image Id and the user Id.

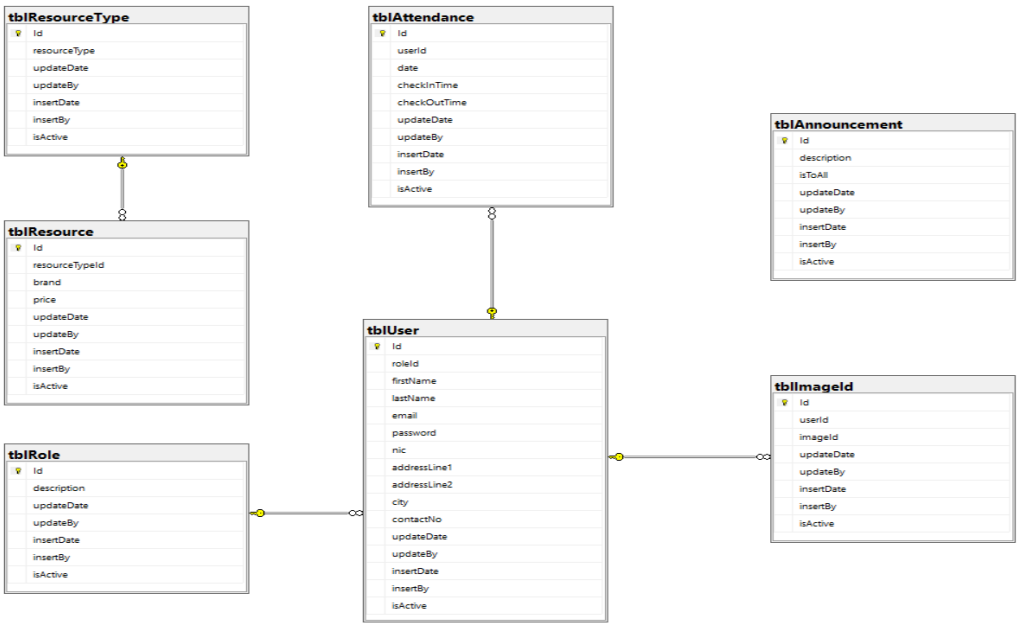

Figure 09: Database Diagram

DELL-I5-TD.SAFELA...dbo.tblAttendance

Id	userId	date	checkInTime	checkOutTi...	updateDate	updateBy	insertDate	insertBy	isActive
1	1	2024-11-12	15:24:32	*NULL*	*NULL*	*NULL*	2024-11-12 ...	1	1
2	2	2024-11-12	15:29:33	*NULL*	*NULL*	*NULL*	2024-11-12 ...	1	1

Figure 10: Attendance records are saved automatically to the database.

UPCOMING WORK

Challenges encountered during completed work such as overfitting, improving the model, dataset creation, new user registration etc. were addressed in above sections of the report.

Some tasks are being developed and those have to be completed. Also, some other tasks are remaining to begin. Below is the summary of those remaining tasks along with the challenges.

- Anti-spoofing algorithm is being developed to ensure that fake or face images which are not live to be correctly identified as fake and improve the accuracy through that. Ensuring the algorithm runs efficiently in real-time, seamlessly integrating the anti-spoofing mechanism with the existing face recognition algorithm will be key challenges here.
- An object detection model is being worked on to verify the necessary safety regulations with PPE kit being detected. Achieving high detection accuracy

across various lighting conditions and angles, ensuring generalization of the model will be the challenges that have to be faced.

- Sensitive data of the database should be secured and encrypted. Here, key challenges will be balancing the encryption and decryption process and securely managing encryption.
- Stored Procedures will be written in MSSQL for CRUD operations.
- Backend development of the mobile/web application will be done using C#.
- API testing will be done.
- Frontend of the mobile/web application will be developed using React.
- The system components should be integrated and tested.

PROJECT STATUS SUMMARY

So far, half of the project is completed, and the remaining tasks has to be completed within the time frame. The face recognition CNN models were built and tested but used a pre-trained facial encoding generator (deep learning-based face_recognition library) to achieve high accuracy, more efficiency, easiness of use and address the issue of new user registration. The face encodings were generated on images and stored into a pickle file. And whenever a face was detected by the camera, face encodings were generated and compared with saved encodings to check whether the person is an authorized user. Database design and integration with the face recognition algorithm was done. The attendance was recorded with the timestamp and user Id if a face match was encountered.

And the above-mentioned upcoming work including developing object detection model for safety regulations, completing and integrating the anti-spoofing algorithm with the face recognition code, ensuring the security of the sensitive information of the database, backend and frontend development of the mobile/web application along with query writing and API testing and system integration testing will be the remaining tasks.

The notebooks, Database script and python files of the work can be accessed through the below link:

https://drive.google.com/file/d/1UAXb1Ip6ymNqz2mOvLlNPOQp0ey-hYBo/view?usp=sharing

REFERENCES

[1] Schroff, F., Kalenichenko, D. and Philbin, J. (2015) 'FaceNet: A unified embedding for face recognition and clustering', 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 815–823. doi:10.1109/cvpr.2015.7298682.

[2] (No date) (PDF) face recognition and face detection benefits and challenges section A-research paper 2561 EUR. Available at: https://www.researchgate.net/publication/372317316_Face_Recognition_and_Face_Detection_Benefits_and_Challenges_Section_A-Research_paper_2561_Eur (Accessed: 15 November 2024).

[3] Deore, M. (2022) FaceNet architecture, Medium. Available at: https://medium.com/analytics-vidhya/facenet-architecture-part-1-a062d5d918a1 (Accessed: 15 November 2024).

[4] GeeksforGeeks (2022) FaceNet - using facial recognition system, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/facenet-using-facial-recognition-system/ (Accessed: 15 November 2024).

[5] Wang, C.-F. (2018) How does a face detection program work? (using neural networks), Medium. Available at: https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff (Accessed: 15 November 2024).

[6] Your machine learning and Data Science Community (no date a) Kaggle. Available at: https://www.kaggle.com/ (Accessed: 15 November 2024).

[7] Hugging face – the AI community building the future. (no date) Hugging Face –. Available at: https://huggingface.co/ (Accessed: 15 November 2024).

[8] API documentation : tensorflow V2.16.1 (no date) TensorFlow. Available at: https://www.tensorflow.org/api_docs (Accessed: 15 November 2024).

[9] Geitgey, A. (2020) Machine learning is fun! part 4: Modern face recognition with deep learning, Medium. Available at: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78 (Accessed: 15 November 2024).

[10] Face recognition¶ (no date) Face Recognition - Face Recognition 1.4.0 documentation. Available at: https://face-recognition.readthedocs.io/en/latest/readme.html (Accessed: 15 November 2024).