

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY



GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

FACIAL RECOGNITION USING THE
FACENET MODEL

Student: Lang Truong An
Student ID: B1910609
Class: 2019-2023 (K45)
Advisor: Associate Prof. Dr. Nguyen Thai Nghe

Can Tho, 12/2023

**CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY**



**GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)**

**FACIAL RECOGNITION USING THE
FACENET MODEL**

**Student: Lang Truong An
Student ID: B1910609
Class: 2019-2023 (Cohort K45)
Advisor: Associate Prof. Dr. Nguyen Thai Nghe**

Can Tho, 12/2023

TRƯỜNG ĐẠI HỌC CẦN THƠ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT
NAM

TRƯỜNG CÔNG NGHỆ THÔNG
TIN VÀ TRUYỀN THÔNG

Độc lập – Tự do – Hạnh phúc

**XÁC NHẬN CHỈNH SỬA LUẬN VĂN THEO YÊU
CẦU CỦA HỘI ĐỒNG**

Tên đề tài: Nhận dạng gương mặt với mô hình Facenet - Facial recognition
using the Facenet model.

Họ tên sinh viên: Lãng Trường An

MASV: B1910609

Mã lớp: DI19V7F1

Đã báo cáo tại hội đồng khoa: Công Nghệ Thông Tin

Ngày báo cáo: 04/12/2023.

Hội đồng báo cáo gồm:

1.TS. Trần Công Ân

Chủ tịch hội đồng

2.TS. Bùi Võ Quốc Bảo

Ủy viên

3.PSG.TS. Nguyễn Thái Nghe

Thư ký

Luận văn đã được chỉnh sửa theo góp ý của Hội đồng.

Cần Thơ, ngày tháng năm 2023

Giảng viên hướng dẫn

(Chữ ký của giảng viên)

PSG.TS. Nguyễn Thái Nghe

ADVISOR'S FEEDBACK



.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Cần Thơ, ngày tháng năm 2023

Giảng viên hướng dẫn

(Chữ ký của giảng viên)

PSG.TS. Nguyễn Thái Nghe

ACKNOWLEDGMENTS

First and foremost, I extend my heartfelt gratitude to all the teachers at the School of Information Technology and Communication, particularly those in the Department of Information Technology, for their dedicated guidance and invaluable knowledge imparted during my study and training at the school.

Next, I express sincere thanks to Associate Prof. Dr. Nguyen Thai Nghe for wholeheartedly assisting me, providing invaluable feedback, and creating optimal conditions throughout the process of completing my thesis. His support was instrumental in successfully finishing my graduation thesis. Additionally, I extend my gratitude to my advisor, Mr. Tran Minh Tan, whose unwavering support, encouragement, and guidance have been invaluable throughout my academic journey.

With utmost gratitude, I especially thank my grandparents, parents, and all my relatives for creating an optimal environment during my educational journey. Their unwavering support provided me the peace of mind to pursue my studies, allowing me to confidently complete my university journey and graduation thesis while developing myself.

Throughout my thesis work, despite my dedication, I encountered limitations in my knowledge and practical experience, leading to some research shortcomings. I sincerely hope for your understanding and valuable feedback, dear teachers, to leverage your experience and wisdom, enhancing the quality of my work and broadening my knowledge.

Finally, I respectfully extend my best wishes to all esteemed teachers, wishing you to always maintain your fervent passion in nurturing and educating the younger generation. Once again, I wish you abundant health, joy, and continuous success in your careers.

Can Tho, November 15, 2022

Author

Lang Truong An

TÓM LƯỢC

Bài luận này tập trung vào việc nhận diện khuôn mặt, tập trung vào việc triển khai mô hình Facenet. Vấn đề nghiên cứu tập trung vào hiệu suất của mô hình Facenet trong việc nhận diện và xác định khuôn mặt một cách chính xác trong các điều kiện và môi trường khác nhau. Theo phương pháp học thuật, nghiên cứu sử dụng các kỹ thuật học sâu và mạng nơ-ron để huấn luyện và đánh giá hiệu suất của mô hình Facenet trong các nhiệm vụ nhận diện khuôn mặt.

Kết quả chính cho thấy độ chính xác đáng kể của mô hình trong việc nhận diện khuôn mặt trên bộ dữ liệu cá nhân mà tôi thu thập và dưới các điều kiện môi trường khác nhau. Hơn nữa, luận văn bàn luận về những điểm mạnh của mô hình trong xử lý các biến thể về góc nhìn, ánh sáng và biểu cảm khuôn mặt, góp phần làm tăng tính mạnh mẽ của nó.

Tóm lại, nghiên cứu nhấn mạnh vào hiệu quả của mô hình Facenet trong các nhiệm vụ nhận diện khuôn mặt, nhấn mạnh vào tiềm năng ứng dụng của nó trong các hệ thống an ninh, giám sát và xác định cá nhân.

ABSTRACT

This thesis delves into the realm of facial recognition, focusing on the implementation of the Facenet model. The research problem addresses the efficacy of the Facenet model in accurately recognizing and identifying faces in various settings and conditions. Methodologically, the study employs deep learning techniques and neural networks to train and evaluate the Facenet model's performance in facial recognition tasks.

The main results show the model's commendable accuracy in identifying faces on the personal dataset I collected and under different environmental conditions. Furthermore, the thesis discusses the model's strengths in handling variations in pose, illumination, and facial expressions, contributing to its robustness.

In conclusion, the study underscores the effectiveness of the Facenet model in facial recognition tasks, highlighting its potential applications in security, surveillance, and personal identification systems.

LIST OF CONTENT

LIST OF CONTENT	i
LIST OF TABLES	iii
LIST OF FIGURES	iv
LIST OF ABBREVIATIONS	vii
CHAPTER 1: INTRODUCTION	1
1.1 The Purpose of the Study	1
1.2 The Problem of the Study	1
1.3 Related Work	1
1.4 Thesis and General Approach	2
1.4.1 Thesis Statement	2
1.4.2 General Approach	2
1.5 Criteria for Study's Success	3
1.6 Outline of the Thesis	3
CHAPTER 2: LITERATURE REVIEW	4
2.1 Overview of Facial Recognition	4
2.2 Existing Models and Techniques in Facial Recognition	4
2.3 The FaceNet Model – Architecture	4
2.3.1 FaceNet Architecture:	5
2.3.2 Triplet Loss	6
2.3.3 FaceNet Variations	7
2.3.4 Conclusion	8
2.4 Features and Advantages of Facenet Model	8
2.4.1 Key Features:	8
2.4.2 Advantages	8
2.4.3 Overall Impact:	9
2.5 The MTCNN Model in face recognition	9
2.6 The SVM Model in face recognition	11
2.7 Haar Cascade	11
2.8 Deployment environment and technology	12
CHAPTER 3: METHODOLOGY	13
3.1 Data Collection	13
3.2 Preprocessing data	13
3.3 Face Detection	15
3.4 Automate the preprocessing	17

3.5	Feature extraction (Embedding).....	22
3.6	Train the Support Vector Machine (SVM) model	24
3.6.1	Label Encoding	24
3.6.2	Plotting a Graph	24
3.6.3	Data Splitting	25
3.6.4	Training an SVM Model.....	26
3.6.5	Prediction on Training and Testing Data	26
3.6.6	Calculating Accuracy	27
CHAPTER 4: DEPLOYING REAL-TIME FACIAL RECOGNITION MODEL.....		28
4.1	Initializing Libraries and Models	28
4.2	Face Processing and Recognition.....	29
4.3	Calculating Face Detection Speed	30
4.4	Displaying the Video stream and Exiting the Program.....	31
4.5	Result.....	31
CHAPTER 5: TESTING ON TEST SET, COMPARISON, EVALUATION.....		37
5.1	Testing on Test set	37
5.2	Comparison with Resnet model	43
5.2.1	Deploy the Resnet model.....	43
5.2.2	Compare the Facenet model with the Resnet model.....	48
5.3	Evaluation	48
CHAPTER 6: CONCLUSION		50
6.1	Summary of Achievements	50
6.2	Limitations of the model	50
6.3	Proposed Future Development Directions	50

LIST OF TABLES

Table 1. All information about recognition accuracy and FPS part 1.	36
Table 2. All information about recognition accuracy and FPS part 2.	36
Table 3. Evaluate the model on the training and test set.....	48
Table 4. The accuracy of the prediction model on the Test set	48
Table 5. Average processing time per image in Test set	48
Table 6. Average accuracy of correctly predicted images	48

LIST OF FIGURES

Image 1. FaceNet Architecture Diagram	5
Image 2. FaceNet Triplet Loss	6
Image 3. Triplet Selection	7
Image 4. MTCNN: Stage architecture of the model used for face detection and landmark extraction.....	10
Image 5. installing and importing necessary libraries.....	14
Image 6. Read image and show image.....	14
Image 7. Face detection.....	15
Image 8. The results of face detection and annotating the detected face on the image	16
Image 9. Face Cropping	17
Image 10. Initializes the FACELOADING object.....	17
Image 11. Function to extract a face from an image file.	18
Image 12. Loads all the face images from a given directory.	19
Image 13. Loads face images from all directories (each directory represents a class/label).....	20
Image 14. Extract faces and corresponding labels in the train set	20
Image 15. Print face extraction information in the dataset	21
Image 16. Prints a list of images for which faces could not be extracted from the dataset.....	21
Image 17. Table to print face extraction information in the dataset	21
Image 18. The code prints all face images in the dataset.....	22
Image 19. The facial images in the dataset have been preprocessed	22
Image 20. Face embedding function	22
Image 21. Calculate embeddings for each face image in the dataset.....	23
Image 22. Save the embeddings.....	23
Image 23. Transform labels (Y) into integer form.....	24
Image 24. Plots the first element in EMBEDDED_X and assigns the label on the y-axis with the value from Y[0].	24
Image 25. Utilizes train_test_split to divide the data into training (X_train, Y_train) and testing (X_test, Y_test) sets.....	25
Image 26. Uses SVC from sklearn.svm to create a Support Vector Machine model	26

Image 27. Predicts labels for the training data (X_train) and testing data (X_test).	26
Image 28. Calculate the accuracy of the classification model based on the training set	27
Image 29. Calculate the accuracy of the classification model based on the test set.	27
Image 30. Save the model	27
Image 31. Initializing Libraries and Models	28
Image 32. Face Processing	29
Image 33. Recognition	30
Image 34. Face Detection Speed.....	30
Image 35. Displaying the video stream.....	31
Image 36. Face recognition under normal conditions - Not expressive	31
Image 37. Face recognition under normal conditions – Smiling	32
Image 38. Face recognition under normal conditions – Uncomfortable	32
Image 39. Face recognition under normal conditions – Wear glasses.....	33
Image 40. Face recognition under normal conditions – Left-leaning angle	33
Image 41. Face recognition under normal conditions – Right-leaning angle.....	34
Image 42. Face recognition under normal conditions - Upward angle.....	34
Image 43. Face recognition under normal conditions - Downward angle.....	35
Image 44. Face recognition in low light conditions.....	35
Image 45. Face recognition in strong light conditions.....	36
Image 46. Recognize faces and predict the identity of the Test set based on a pre-trained model.....	37
Image 47. The accuracy of the prediction model on the Test set	38
Image 48. Average processing time per image in Test Set.	38
Image 49. Print out the prediction information table for each image	39
Image 50. List of correctly predicted images – part 1	40
Image 51. List of correctly predicted images – part 2	41
Image 52. List of correctly predicted images – part 3	42
Image 53. List of incorrectly predicted images.....	42
Image 54. Average accuracy of correctly predicted images.	43
Image 55. Import library for Resnet model.....	43
Image 56. Load the ResNet model.....	44

Image 57. Initialize the FACELOADING object for the Resnet model.....	44
Image 58. Load images and labels from dataset of the Resnet model	44
Image 59. Image preprocessing and embedding into feature space using ResNet model.....	45
Image 60. Evaluate the model on the training and test sets of the Resnet model.....	45
Image 61. The accuracy of the prediction Resnet model on the Test set.....	45
Image 62. Average processing time per image in Test Set of the Resnet model.....	46
Image 63. List of correctly predicted images of the Resnet model.	46
Image 64. List of incorrectly predicted images of the Resnet model	47
Image 65. Average accuracy of correctly predicted images of the Resnet model....	47

LIST OF ABBREVIATIONS

ABBREVIATION	DEFINITION
MTCNN	Multi-task Cascaded Convolutional Networks
SVM	Support Vector Machine
CNNs	Convolutional Neural Networks
LBP	Local Binary Patterns
ResNet	Residual Network

CHAPTER 1: INTRODUCTION

1.1 The Purpose of the Study

The goal of this study is to implement the Facenet model to provide an objective assessment of its performance, accuracy, and processing speed. The research focuses on building an automated pipeline to extract important facial attributes, convert them into embedding vectors using the Facenet model, and use machine learning methods such as SVM for classification and facial recognition in real-life situations. To ensure a comprehensive evaluation of the model's performance, accuracy, and processing speed, the study includes evaluations on a test dataset, implementation of a real-time camera recognition model, and comparison between Facenet and Resnet models.

1.2 The Problem of the Study

Today, facial recognition has gained popularity due to its myriad applications across various domains such as security, surveillance, commercial ventures, healthcare, education, and service industries. Consequently, numerous facial recognition models have been researched and developed to cater to societal needs.

However, despite its vast potential and extensive applications, facial recognition also presents certain challenges. These include algorithmic accuracy issues, variations in lighting conditions and camera angles, differences in individual facial features, and notably, the processing efficiency and speed constraints in real-time applications.

The development of "Facial recognition using the Facenet model" aims to address challenges related to performance, accuracy, processing speed, variations in lighting conditions, and individual facial characteristics. This holds promise as a potential model in the field of facial recognition.

1.3 Related Work

In the field of facial recognition, numerous prior studies have focused on applying machine learning models and feature extraction techniques to improve the accuracy and performance of recognition systems. Traditional methods such as Local Binary Patterns (LBP) and Histogram of Oriented Gradients (HOG) have been used to extract features from faces; however, they often encounter limitations in dealing with diverse variations in lighting and angles of capture.

Moreover, deeper models like Convolutional Neural Networks (CNNs) have introduced a new approach, enabling the learning of complex features from image data, thus enhancing facial recognition capabilities under diverse conditions. The Facenet model, in particular, has gained significant attention for its ability to generate high-detail embedding vectors for faces, thereby boosting the accuracy of facial recognition.

Hence, "Facial recognition using the Facenet model" has been researched and developed by combining the Facenet model with machine learning algorithms like

Support Vector Machines (SVM) to create facial recognition systems capable of accurate and efficient real-time predictions.

1.4 Thesis and General Approach

1.4.1 Thesis Statement

The main goal of the thesis "Face recognition using the Facenet model" is to develop a highly accurate face recognition model under different conditions. This is achieved by using the MTCNN model for face recognition, using the Facenet model for facial feature extraction, and integrating Support Vector Machine (SVM) for recognition.

1.4.2 General Approach

Research documents

Research the Facenet model

Data collection and preprocessing

- Collect diverse datasets of facial images in different lighting conditions, angles and environments.
- Pre-processing of collected data ensures uniformity and quality for model training.

Model development

- Prepare training data
- Face Detection
- Automate the preprocessing
- Feature extraction (Embedding)
- Train the Support Vector Machine (SVM) model
 - Prediction on Training and Testing Data

Deploying Real-Time Facial Recognition Model

- Deploy the built model into real scenarios to evaluate practical applicability.
- Analyze system performance under real-world conditions and measure computational efficiency in terms of accuracy, measuring image processing Frames Per Second (FPS).

Testing on test set, Comparison with Resnet model, Evaluation

- Testing on test set
- Comparison with Resnet model
- Evaluate model performance on test datasets using various metrics such as model accuracy, individual image accuracy.

- Processing speed

1.5 Criteria for Study's Success

Criteria to evaluate the success of research on "Facial recognition using the Facenet model" may include:

Accuracy: Determines the percentage of faces correctly recognized in the test data set.

Robustness: Test the ability to recognize faces under many different conditions such as changes in lighting, posture, and facial expressions.

Processing Speed: Evaluates the model's performance in image processing and face recognition, especially when applied in real time.

1.6 Outline of the Thesis

CHAPTER 1: Introduction: Purpose of the Study, Problem of the Study, Related work – cite previous work, Thesis and General Approach, Criteria for Study's Success, Outline of the Thesis.

CHAPTER 2: Literature Review: Overview of Facial Recognition, Existing Models and Techniques in Facial Recognition, The FaceNet Model - Architecture, Features and Advantages of Facenet Model, The MTCNN Model in face recognition, The SVM Model in face recognition, Haar Cascade, Deployment environment and technology.

CHAPTER 3: Methodology: Data Collection, Preprocessing data, Face Detection, Automate the preprocessing, Feature extraction (Embedding), Train the Support Vector Machine (SVM) model.

CHAPTER 4: Deploying Real-Time Facial Recognition Model.

CHAPTER 5: Testing on the Test set, Comparison with Resnet model, Evaluation.

CHAPTER 6: Conclusion: Summarize the results achieved and proposed future development directions.

References

CHAPTER 2: LITERATURE REVIEW

2.1 Overview of Facial Recognition

Facial recognition is a way of identifying or confirming an individual's identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time.

Facial recognition is a category of biometric security. Other forms of biometric software include voice recognition, fingerprint recognition, and eye retina or iris recognition. The technology is mostly used for security and law enforcement, though there is increasing interest in other areas of use.

2.2 Existing Models and Techniques in Facial Recognition

There are various models and techniques employed in facial recognition, some of which include:

Convolutional Neural Networks (CNNs): CNNs are commonly used for facial recognition due to their ability to extract features from images effectively. Architectures like VGG, ResNet, and Inception have been adapted and fine-tuned for facial recognition tasks.

Eigenfaces: This technique uses principal component analysis (PCA) to reduce the dimensionality of face images. It represents faces as a linear combination of basic patterns known as eigenfaces.

Local Binary Patterns (LBP): LBP encodes local texture information in an image and has been used effectively for facial feature extraction and recognition.

DeepFace: Developed by Facebook, DeepFace utilizes a deep learning architecture to identify faces in images with high accuracy by mapping facial features into a multi-layer neural network.

Dlib: Dlib is a popular open-source library containing machine learning algorithms, including facial recognition functionality. It provides pre-trained models and tools for facial landmark detection and recognition.

Facial Landmark Detection: Techniques detecting key points on a face (like eyes, nose, mouth) are often used as a precursor to facial recognition. Models like the 68-point landmark detector from dlib or the 5-point landmark detector used in mobile applications fall under this category.

In addition to the models and techniques listed above there is still the **Facenet model**. This model by Google uses a triplet loss function to learn a mapping of face images into a high-dimensional space where distances directly correspond to a measure of face similarity.

2.3 The FaceNet Model – Architecture

FaceNet is a neural network-based system that uses deep learning algorithms to recognize faces. Developed by Google researchers in 2015, the system uses a

convolutional neural network (CNN) to map facial features into a Euclidean space where distances directly correspond to a measure of face similarity. This space is then used to compare faces and recognize whether they belong to the same person or not.

2.3.1 FaceNet Architecture:

FaceNet uses a deep network that learns and extracts various facial features. These features are then mapped to a 128-dimensional space, where images belonging to the same person are close to each other and far from images belonging to different subjects.



Image 1. FaceNet Architecture Diagram

Deep neural network: originally, FaceNet used Inception Network as its backbone architecture. The core concept of Inception is the use of 1X1 filters for dimensionality reduction (for example converting a 256x256x3 RGB image to a 256x256x1 image). Other filters (e.g., 3x3) can be used for the same input, combining the different outputs (this technique is called “inception module”). Nowadays, a multitude of backbone networks can be used in place of Inception, with ResNet being the most popular choice.

L2 Normalization: the outputs are normalized using L2 Norm, also known as the Euclidean norm. It’s the distance of the output from the origin in the n-dimensional space and it is computed by the square root of the sum of the squared vector values.

Embeddings: the embeddings are calculated and mapped in the relevant feature space. The embeddings can be seen as a “summary” of an identity. Using such a smaller representation (compared to the full image) is crucial for tackling the face recognition task effectively.

Loss Function: this is one of the differentiating features of FaceNet, especially when compared to standard Siamese architectures. Hold your breath, the next paragraph is dedicated solely to FaceNet's loss function: i.e. triplet loss.

2.3.2 Triplet Loss

The triplet loss function is the key to training the FaceNet model. The loss function compares the distance between an anchor image and a positive image of the same person with the distance between the anchor image and a negative image of a different person. The goal is to minimize the distance between the anchor image and the positive image while maximizing the distance between the anchor image and the negative image.

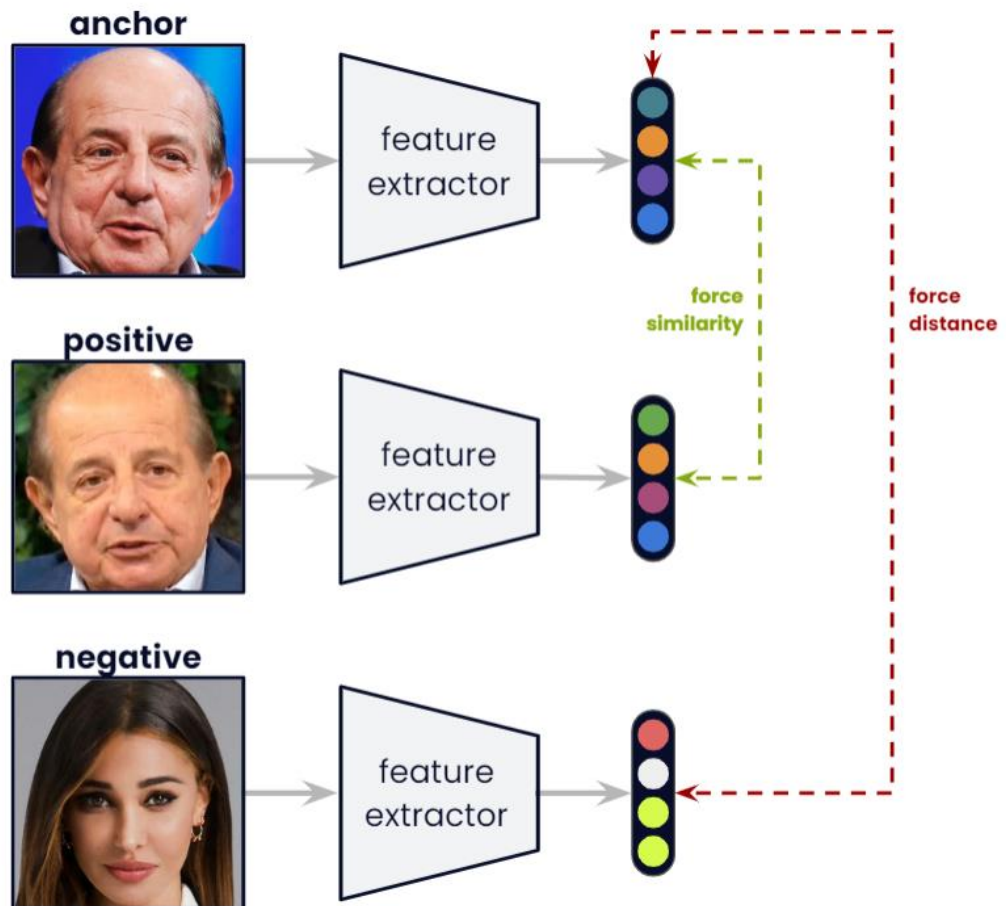


Image 2. FaceNet Triplet Losses

The selection of triplets (the three pictures to be given as input to the function) for the triplet loss function is crucial for the effectiveness of the training. The triplets should be carefully selected to ensure that the network learns to produce highly discriminative features for facial recognition. One approach is to use semi-hard triplets, which are triplets where the negative is farther from the anchor than the positive, but closer than the positive plus an arbitrary margin.

$$d(a, p) < d(a, n) < d(a, p) + \text{margin}$$

Another approach is to use hard triplets, which are triplets where the negative image is closer to the anchor image than the positive image.

$$d(a,n) < d(a,p)$$

Hard triplets are more challenging to learn from but can be useful for improving the robustness of the network to variations in lighting, pose, and expression. The selection of triplets is an important aspect of training the FaceNet architecture and requires the balancing of desired performance with the available resources (computing power & time).



Image 3. Triplet Selection

2.3.3 FaceNet Variations

Since the original development of FaceNet, several variations of the architecture have been proposed. One such variation is the MobileFaceNet, which is designed to work with fewer parameters, for mobile devices with limited computational resources.

Another variation is the SphereFace, which uses a different loss function than the triplet loss. The SphereFace loss function aims to maximize the cosine similarity between the features of the anchor and positive images while minimizing the cosine similarity between the features of the anchor and negative images. The objective of SphereFace is to improve face recognition

performance for open-set use cases: namely, those scenarios where the people to identify in production are not represented in your training data.

2.3.4 Conclusion

Facial recognition technology has come a long way since its inception, and FaceNet is a significant breakthrough in this field. The FaceNet architecture, with its triplet loss function, has proven to be highly accurate in recognizing and verifying faces. With variations like MobileFaceNet and SphereFace, FaceNet has become more versatile and adaptable to different use cases.

2.4 Features and Advantages of Facenet Model

2.4.1 Key Features:

High-Quality Embeddings: FaceNet generates high-dimensional embeddings for face images that encapsulate discriminative facial features, making them well-suited for face recognition tasks.

End-to-End Learning: It employs an end-to-end learning approach, eliminating the need for separate feature extraction or engineering stages. The model directly learns to produce optimized facial embeddings.

Triplet Loss Function: The utilization of the triplet loss function during training ensures that the model learns to map similar faces closer together and dissimilar faces farther apart in the embedding space, enhancing discrimination capability.

Siamese Network Architecture: The use of a Siamese neural network design allows for simultaneous processing of pairs of face images, facilitating the extraction of embeddings for both images.

Robustness to Variations: FaceNet exhibits robustness to variations in facial expressions, poses, lighting conditions, and other environmental factors due to its ability to capture hierarchical and abstract facial features.

2.4.2 Advantages

State-of-the-Art Performance: FaceNet achieved state-of-the-art performance in face recognition benchmarks, surpassing prior methods and setting new standards in accuracy and efficiency.

Generalization Across Diverse Conditions: Trained on extensive datasets, FaceNet's embeddings generalize well across diverse facial variations, allowing it to recognize faces accurately in various real-world scenarios.

Effective Face Verification: Its embeddings enable efficient face verification by measuring distances or similarities between embeddings, making it valuable for applications like access control and security systems.

Elimination of Manual Feature Engineering: FaceNet's ability to learn representations directly from raw data eliminates the need for labor-intensive manual feature engineering, streamlining the development of face recognition systems.

Applicability to Real-World Scenarios: The robustness and accuracy of FaceNet make it applicable in a wide range of practical scenarios, including surveillance, personalization, and identity verification across industries.

2.4.3 Overall Impact:

FaceNet's remarkable features and advantages have significantly advanced the field of face recognition technology. Its ability to generate high-quality facial embeddings, coupled with robustness and accuracy across various conditions, has made it a pivotal tool in biometric identification systems and has opened doors to numerous applications requiring reliable and efficient face recognition capabilities.

2.5 The MTCNN Model in face recognition

The MTCNN (Multi-task Cascaded Convolutional Neural Network) is a deep learning-based face detection model designed for accurate and efficient detection of faces within images. It comprises three stages of neural networks, each responsible for different tasks in the face detection process: face detection, facial landmark localization, and bounding box regression.

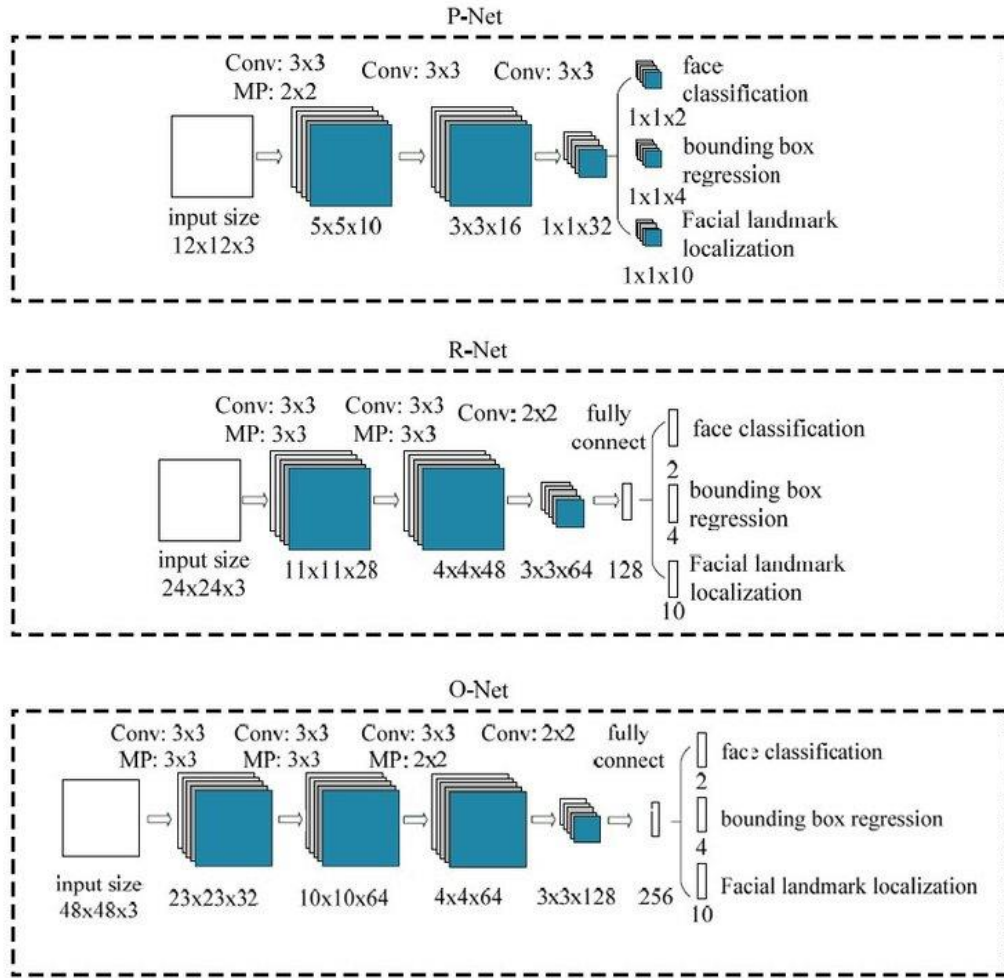


Image 4. MTCNN: Stage architecture of the model used for face detection and landmark extraction.

Components and Functionality:

Stage 1: Proposal Network (P-Net):

The P-Net is the initial stage responsible for generating candidate bounding boxes likely to contain faces.

It uses a fully convolutional network to propose potential face regions by sliding a small window over the input image and classifying each window as containing a face or not.

This stage generates multiple candidate regions of interest (ROIs) with different scales and positions.

Stage 2: Refinement Network (R-Net):

The R-Net refines the face candidates generated by the P-Net.

It filters out false positives and improves the quality of the bounding box proposals by further classifying and regressing the bounding boxes.

This stage helps in reducing the number of false positives and refining the bounding boxes around detected faces.

Stage 3: Output Network (O-Net):

The O-Net is the final stage that performs more precise facial feature localization and bounding box regression.

It conducts facial landmark detection, identifying key points such as eyes, nose, and mouth within the refined bounding boxes.

Additionally, it further refines the bounding box coordinates and filters out overlapping or inaccurate detections.

2.6 The SVM Model in face recognition

Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. The main idea behind SVMs is to find a hyperplane that maximally separates the different classes in the training data. This is done by finding the hyperplane that has the largest margin, which is defined as the distance between the hyperplane and the closest data points from each class. Once the hyperplane is determined, new data can be classified by determining on which side of the hyperplane it falls. SVMs are particularly useful when the data has many features, or when there is a clear margin of separation in the data.

In this thesis topic, the SVM model is employed to classify facial embeddings generated by FaceNet. Specifically, after FaceNet produces facial embeddings, which are numerical representations of facial features, SVM can be utilized for classification and identification purposes. Each facial embedding corresponds to an individual or belongs to a specific class, and SVM is employed to classify and determine the identity associated with each facial embedding.

SVM is trained on labeled facial embeddings from the training dataset to learn how to distinguish between different individuals. Once trained, SVM demonstrates the capability to effectively classify and identify the person represented in each facial embedding.

2.7 Haar Cascade

Haar Cascade is a machine learning-based object detection method used for identifying objects in images or video frames. It's based on the Haar wavelet technique that uses a set of simple features called Haar-like features.

The Haar Cascade classifier is trained using positive and negative images. Positive images contain the object to be detected (e.g., faces), while negative images contain backgrounds without the object. The classifier is trained to understand the difference between these positive and negative samples by learning features that are distinctive to the object being detected.

The Haar-like features used in this method are rectangular filters that are applied to regions of an image. These features can capture edge, line, and texture information from the image by calculating the difference between the sums of pixel intensities in specific regions. For instance, one feature might focus on the contrast between the pixels in the area of the eyes and the surrounding cheeks.

The concept of cascades in Haar Cascades refers to a series of classifiers arranged in a hierarchy. Each stage in the cascade removes false positives progressively by applying more complex classifiers. This hierarchical approach allows for faster processing of regions that are unlikely to contain the target object, enhancing the detection speed.

Haar Cascade classifiers are commonly used in computer vision for various tasks such as face detection, object recognition, pedestrian detection, and more, due to their effectiveness in detecting objects within images or video frames.

2.8 Deployment environment and technology

Deployment Environment:

Google Colab: Utilizing the Colab environment for code execution, data storage, and accessing Google Drive.

PyCharm: A popular integrated development environment (IDE) used to deploy real-time facial recognition model.

Technologies:

Python: The primary programming language for scripting.

OpenCV (cv2): Image and video processing library.

NumPy: Library for array manipulation, matrices, and scientific computation.

TensorFlow (tf): Machine learning and deep learning library.

Matplotlib (plt): Visualization library used for displaying images.

MTCNN (Multi-Task Cascaded Convolutional Networks): Employed for face detection within images.

Keras-FaceNet: Utilizing FaceNet, a deep learning model for face embedding and feature extraction.

Scikit-learn: Machine learning library used for training and employing SVM (Support Vector Machine) for face classification.

PrettyTable: Used for displaying data in a readable tabular format.

Pickle: Utilized for saving and loading Python objects.

Warnings: A library for managing and suppressing unnecessary warnings.

CHAPTER 3: METHODOLOGY

3.1 Data Collection

Data collection and pre-processing play an important role in building an effective facial recognition system.

Data collection requirements:

- Photo must clearly show the face
- Photo must have only one person
- Must have supplier's consent
- Diverse expressions and poses.
- Diverse lighting conditions

The total number of images obtained is 400, from 6 different people, including:

- Thi Dong with 40 images.
- Dang Tam with 40 images.
- Hoang Hao with 40 images.
- Quoc Vinh with 40 images.
- Bich Huyen with 40 images.
- Truong An with 200 images.

3.2 Preprocessing data

The image count for each person will be divided into two parts following a 3:1 ratio. The portion aligned with the 3 ratio will be used as the dataset, while the remaining part will serve as the test set.

Consequently:

- The dataset set contains 300 images.
- The test set comprises 100 images.

Within the dataset, images are organized into 6 folders, each named after an individual. These folders contain the images intended for the dataset.

The test set incorporates all images of the 6 individuals, and each person's images are stored in the format of [name number]. For instance: Truong An 1.jpg, Thi Dong 5.jpg.

```
!pip install mtcnn
import cv2 as cv
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Image 5. installing and importing necessary libraries

Prepares the environment by installing the 'mtcnn' package (essential for facial detection), importing necessary libraries, and configuring the environment to suppress warning messages from TensorFlow.

```
[25] img = cv.imread("/content/drive/MyDrive/face_recognition_Facenet/dataset/Truong An/2.jpg")
# opencv BGR channel format and plt reads images as RGB channel format
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img) # RGB
```

<matplotlib.image.AxesImage at 0x78ed3cbc0670>

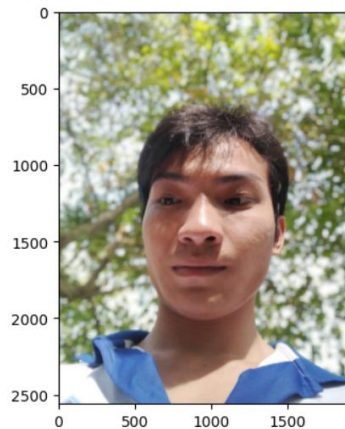


Image 6. Read image and show image

Uses OpenCV (cv) to read an image file

cv.cvtColor(img, cv.COLOR_BGR2RGB): Converts the color channels of the image from BGR format (used by OpenCV) to RGB format.

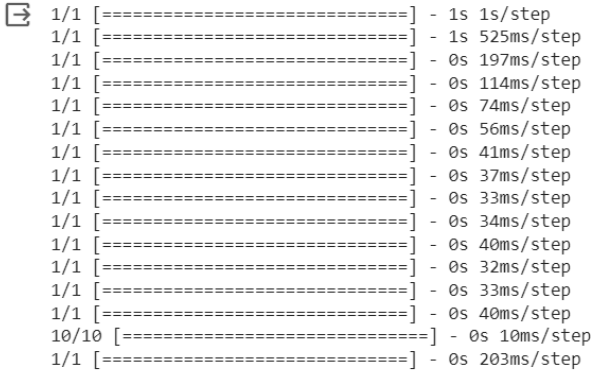
This is necessary because OpenCV reads images in BGR format by default, while plt.imshow() from Matplotlib expects images in RGB format for display.

plt.imshow(img): Displays the image using Matplotlib. The image (img) is in RGB format after the conversion, allowing Matplotlib to show it correctly.

3.3 Face Detection

```
from mtcnn.mtcnn import MTCNN

detector = MTCNN()
results = detector.detect_faces(img)
```



```
1/1 [=====] - 1s 1s/step
1/1 [=====] - 1s 525ms/step
1/1 [=====] - 0s 197ms/step
1/1 [=====] - 0s 114ms/step
1/1 [=====] - 0s 74ms/step
1/1 [=====] - 0s 56ms/step
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 33ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 40ms/step
1/1 [=====] - 0s 32ms/step
1/1 [=====] - 0s 33ms/step
1/1 [=====] - 0s 40ms/step
10/10 [=====] - 0s 10ms/step
1/1 [=====] - 0s 203ms/step
```

Image 7. Face detection

Imports the MTCNN class from the mtcnn module. MTCNN is a neural network-based face detection algorithm that employs a cascaded structure of convolutional networks to identify faces in images.

detector = MTCNN(): Initializes an instance of the MTCNN detector.

results = detector.detect_faces(img): Utilizes the MTCNN detector to detect faces in the provided image (img). The detect_faces method returns a list of face detections, each containing information like the bounding box coordinates, confidence level, and facial landmarks if available.

```
[31] results
     x,y,w,h = results[0]['box']
     img = cv.rectangle(img, (x,y), (x+w, y+h), (0,0,255), (30))
     plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x78ed4453e050>

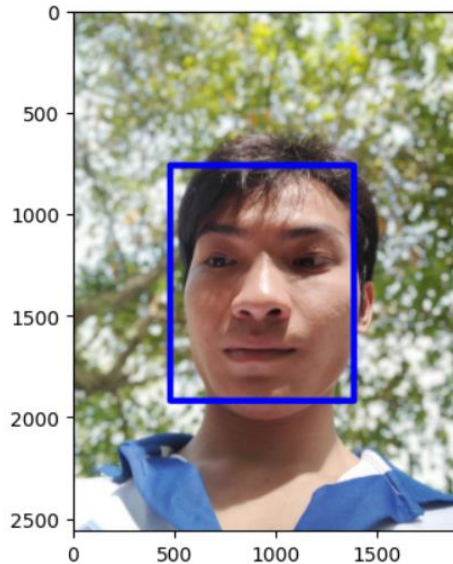


Image 8. The results of face detection and annotating the detected face on the image

x, y, w, h = results[0]['box']: Extracts the coordinates (x, y) of the top-left corner and the width (w) and height (h) of the bounding box of the first detected face. The 'box' key in results[0] holds this information.

img = cv.rectangle(img, (x,y), (x+w, y+h), (0,0,255), (30)): Utilizes OpenCV's cv.rectangle() function to draw a rectangle around the detected face on the image (img).

The parameters include:

- **img:** The image where the rectangle is drawn.
- **(x, y):** Coordinates of the top-left corner of the rectangle.
- **(x+w, y+h):** Coordinates of the bottom-right corner of the rectangle.
- **(0,0,255):** Specifies the color of the rectangle in BGR format (here, it represents red because the order is blue, green, red).
- **(30):** Specifies the thickness of the rectangle border.

```

▶ my_face = img[y:y+h, x:x+w]
  #Facenet takes as input 160x160
  my_face = cv.resize(my_face, (160,160))
  plt.imshow(my_face)

```

↳ <matplotlib.image.AxesImage at 0x78ed3c405f00>

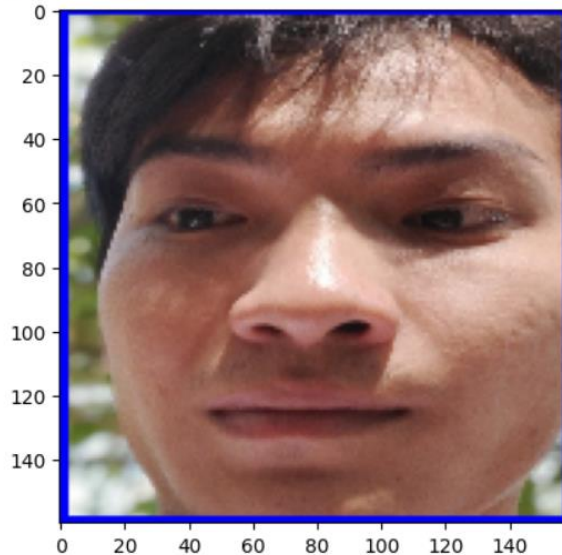


Image 9. Face Cropping

my_face = img[y:y+h, x:x+w]: Extracts the region of interest (ROI), the detected face area, from the original image (img) based on the coordinates obtained from the face detection results.

my_face = cv.resize(my_face, (160,160)): Resizes the extracted face region (my_face) to a size of 160x160 pixels.

plt.imshow(my_face): Uses Matplotlib to display the resized face image. This image has been cropped and resized specifically for compatibility with the FaceNet model, which typically requires input face images of a certain size (in this case, 160x160 pixels).

3.4 Automate the preprocessing

```

▶ from mtcnn.mtcnn import MTCNN
  class FACELOADING:
      def __init__(self, directory):
          self.directory = directory
          self.target_size = (160, 160)
          self.X = []
          self.Y = []
          self.detector = MTCNN()
          # List to save information about the number of extracted images
          self.extracted_info = []
          self.unextracted_images = []

```

Image 10. Initializes the FACELOADING object

__init__(self, directory): Initializes the FACELOADING object.

- **directory:** Sets the directory path where the images are located.
- **target_size:** Sets the desired size for face images after resizing (160x160).
- **X:** List to store extracted face images.
- **Y:** List to store corresponding labels/classes for the images.
- **detector:** Initializes an instance of the MTCNN (Multi-task Cascaded Convolutional Networks) for face detection.
- **extracted_info:** Empty list to hold information about the number of successfully extracted faces.
- **unextracted_images:** Empty list to store information about images for which face extraction failed or no faces were detected.

```
def extract_face(self, filename):
    img = cv.imread(filename)
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    try:
        faces = self.detector.detect_faces(img)
        if faces:
            x, y, w, h = faces[0]['box']
            x, y = abs(x), abs(y)
            face = img[y:y + h, x:x + w]
            face_arr = cv.resize(face, self.target_size)

            if face_arr is not None:
                return face_arr, None
            else:
                return None, "Face extraction failed"
        else:
            return None, "No face detected"
    except Exception as e:
        return None, f"Face extraction error: {str(e)}"
```

Image 11. Function to extract a face from an image file.

extract_face(self, filename): Function to extract a face from an image file.

- **filename:** Path to the image file.
- Reads the image, converts it from BGR to RGB, and attempts to detect faces using the MTCNN detector.
- If a face is detected, it crops the face region, resizes it to the target size, and returns the face image array along with a possible error message.
- If no face is detected or an exception occurs during the process, it returns None along with an error message.


```

def load_faces(self, dir):
    total_images = len(os.listdir(dir))
    extracted_faces = 0
    FACES = []
    for im_name in os.listdir(dir):
        try:
            path = os.path.join(dir, im_name)
            single_face, error = self.extract_face(path)
            # Check the return value from extract_face()
            if single_face is not None:
                FACES.append(single_face)
                extracted_faces += 1
            else:
                # Save information about unextracted
                # images to the unextracted_images list
                file_name = os.path.basename(path)
                label_unextracted = os.path.dirname(path)
                self.unextracted_images.append({
                    'Label_unextracted': label_unextracted,
                    'File_name': file_name, 'Error': error})
        except Exception as e:
            pass
    self.extracted_info.append({'Label': dir, 'Total_images': total_images,
                               'Extracted_faces': extracted_faces})

    return FACES

```

Image 12. Loads all the face images from a given directory.

load_faces(self, dir): Function to loads all the face images from a given directory.

- **dir:** Directory path containing the images for a specific class/label.
- Iterates through all the image files in the directory, attempts to extract faces using `extract_face`, and stores the successfully extracted faces in `FACES`.
- Keeps track of the total number of images, the number of successfully extracted faces, and appends this information to `extracted_info`.
- If face extraction fails for any image, it appends information about the unextracted image to `unextracted_images`.
- Returns the list of extracted face images (`FACES`).

```
def load_classes(self):
    for sub_dir in os.listdir(self.directory):
        path = os.path.join(self.directory, sub_dir)
        FACES = self.load_faces(path)
        labels = [sub_dir for _ in range(len(FACES))]
        self.X.extend(FACES)
        self.Y.extend(labels)
    return np.asarray(self.X), np.asarray(self.Y)
```

Image 13. Loads face images from all directories (each directory represents a class/label).

load_classes(self): Loads face images from all directories (each directory represents a class/label).

- Iterates through subdirectories (each representing a class) in the main directory and calls load_faces to load faces from each class.
- Extends self.X with the face images and self.Y with the corresponding class labels.
- Returns the array representation of self.X and self.Y.

```
faceloading = FACELOADING("/content/drive/MyDrive/face_recognition_Facenet/dataset")
X, Y = faceloading.load_classes()

... 1/1 [=====] - 0s 109ms/step
1/1 [=====] - 0s 77ms/step
1/1 [=====] - 0s 59ms/step
1/1 [=====] - 0s 44ms/step
1/1 [=====] - 0s 49ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 40ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 92ms/step
1/1 [=====] - 0s 90ms/step
2/2 [=====] - 0s 31ms/step
1/1 [=====] - 0s 93ms/step
1/1 [=====] - 3s 3s/step
1/1 [=====] - 1s 633ms/step
1/1 [=====] - 0s 479ms/step
1/1 [=====] - 0s 217ms/step
```

Image 14. Extract faces and corresponding labels in the train set

Creates an instance of the FACELOADING class named faceloading, passing the directory path "/content/drive/MyDrive/.../dataset" as an argument.

Then, it calls the load_classes method on this instance, which triggers the extraction and loading of face images from the specified directory, returning the extracted face data (X) and their corresponding labels (Y).

```

▶ from prettytable import PrettyTable
# Table to print face extraction information in the dataset
table = PrettyTable()
table.field_names = ["Label", "Total_images", "Extracted_faces"]

for info in faceloading.extracted_info:
    label = info['Label']
    label_name = label.split('/')[-1]
    total_images = info['Total_images']
    extracted_faces = info['Extracted_faces']
    table.add_row([label_name, total_images, extracted_faces])

print(table)

```

Image 15. Print face extraction information in the dataset

```


# The table prints a list of images for which
# faces could not be extracted from the dataset
table1 = PrettyTable()
table1.field_names = ["Label Unextracted", "File Name", "Error"]

# Check if the list is not empty
if faceloading.unextracted_images:
    for info in faceloading.unextracted_images:
        label_unextracted = info['Label unextracted'].split('/')[-1]
        file_name = info['File_name']
        error = info['Error']
        table1.add_row([label_unextracted, file_name, error])

print("\nList of Unextracted face images:")
print(table1)

```

Image 16. Prints a list of images for which faces could not be extracted from the dataset



Label	Total_images	Extracted_faces
Bich Huyen	30	30
Thi Dong	30	30
Hoang Hao	30	30
Quoc Vinh	30	30
Dang Tam	30	30
Truong An	150	150

Image 17. Table to print face extraction information in the dataset

```
[19] plt.figure(figsize=(60,60))
      for num,image in enumerate(X):
          ncols = 10
          nrows = len(Y)//ncols + 1
          plt.subplot(nrows,ncols,num+1)
          plt.imshow(image)
          plt.axis('off')
```

Image 18. The code prints all face images in the dataset

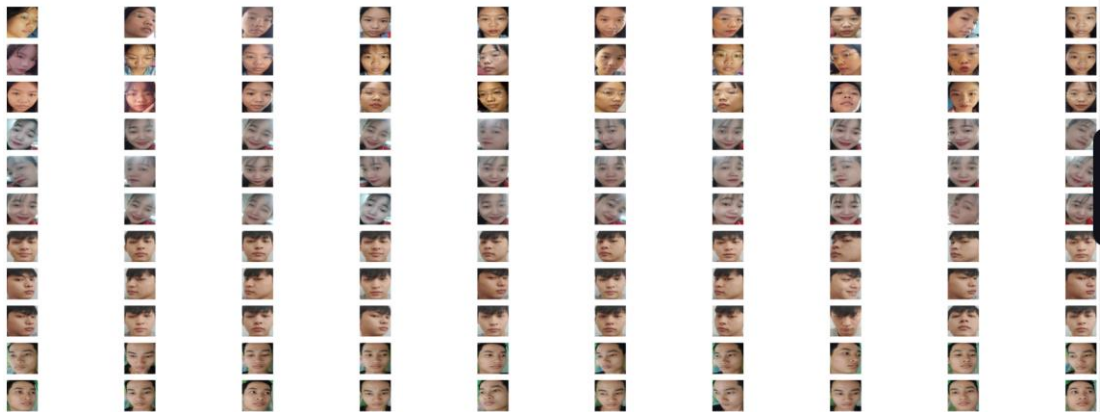


Image 19. The facial images in the dataset have been preprocessed

3.5 Feature extraction (Embedding)

```
!pip install keras-facenet
from keras_facenet import FaceNet
embedder = FaceNet()

def get_embedding(face_img):
    face_img = face_img.astype('float32') # 3D(160x160x3)
    face_img = np.expand_dims(face_img, axis=0)
    # 4D (Nonex160x160x3)
    yhat= embedder.embeddings(face_img)
    return yhat[0] # 512D image (1x1x512)
```

Image 20. Face embedding function

Installs the keras-facenet package

from keras_facenet import FaceNet: Imports the FaceNet class from the keras_facenet library.

embedder = FaceNet(): Initializes an instance of the FaceNet model, which represents a pre-trained FaceNet model capable of generating facial embeddings.

get_embedding: This function receives a face image (face_img) as input and returns its embedding using the FaceNet model.

face_img.astype('float32'): Converts the input face image to a float32 datatype. Neural networks often require input in a specific datatype.

np.expand_dims(face_img, axis=0): Reshapes the image array to a 4D array (None x 160 x 160 x 3). The None dimension is for the batch size, which here is 1, indicating a single image.

yhat = embedder.embeddings(face_img): This line generates the embeddings (a 512-dimensional vector representation) for the input face image using the FaceNet model.

return yhat[0]: Returns the computed embeddings. The embeddings are a 1x1x512 array, but [0] extracts the 512-dimensional vector (representing the face embedding) for further usage.

```
EMBEDDED_X = []

for img in X:
    EMBEDDED_X.append(get_embedding(img))

EMBEDDED_X = np.asarray(EMBEDDED_X)
```




Image 21. Calculate embeddings for each face image in the dataset

Compute embeddings for each face image in the list X using the get_embedding function, then store these embeddings in the EMBEDDED_X list as a NumPy array.

```
save_path = "/content/drive/MyDrive/face_recognition_Facenet/faces_embeddings_done_4classes.npz"
np.savez_compressed(save_path, EMBEDDED_X, Y)
print("File saved successfully.")
```

Image 22. Save the embeddings

Save the embeddings of the faces along with their corresponding labels (in the variable Y) into a compressed file format .npz. It stores the information of face embeddings and their labels into a file located at save_path.

3.6 Train the Support Vector Machine (SVM) model

3.6.1 Label Encoding

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(Y)
Y = encoder.transform(Y)
```

Image 23. Transform labels (Y) into integer form.

- The first line imports the `LabelEncoder` class from the `sklearn.preprocessing` module.
- A new encoder object is created based on the `LabelEncoder` class.
- The `.fit(Y)` method learns to map the labels in the variable `Y` to unique integers. It analyzes and establishes a mapping from string labels to corresponding unique integers.
- The `.transform(Y)` method, after the fit method has learned the mapping, converts the labels in `Y` to corresponding integers based on the learned mapping from fit.
- Labels are replaced with corresponding integer values.

3.6.2 Plotting a Graph

```
plt.plot(EMBEDDED_X[0])
plt.ylabel(Y[0])
```

Text(0, 0.5, '0')

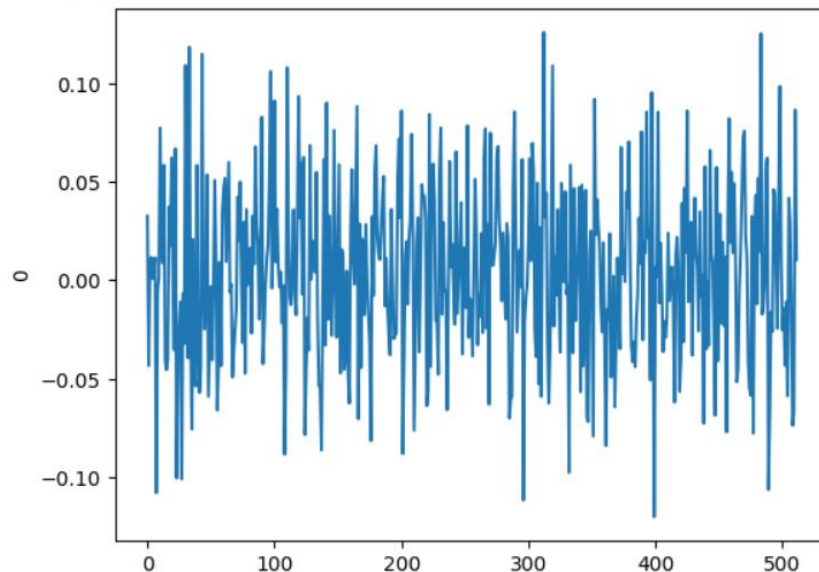


Image 24. Plots the first element in `EMBEDDED_X` and assigns the label on the y-axis with the value from `Y[0]`.

3.6.3 Data Splitting

```
▶ from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(
    EMBEDDED_X, Y, shuffle=True, random_state=17
)
```

Image 25. Utilizes train_test_split to divide the data into training (X_train, Y_train) and testing (X_test, Y_test) sets.

sklearn.model_selection module to split the dataset into two parts: one for training the model (X_train, Y_train) and another for evaluating the model (X_test, Y_test).

EMBEDDED_X: It contains the facial embeddings data where each row represents a facial embedding vector.

Y: This variable contains the corresponding labels for the embedding vectors in EMBEDDED_X.

train_test_split: This function divides the data into training and testing sets based on the provided parameters.

X_train, X_test: These variables store the embedded vectors after being split into training and testing sets.

Y_train, Y_test: These variables contain the corresponding labels for X_train and X_test.

EMBEDDED_X and Y: Data to be split.

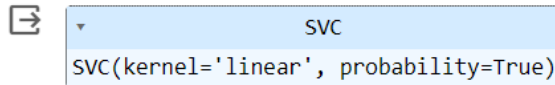
shuffle=True: Allows shuffling of the data before splitting to ensure randomness.

random_state=6: Sets the seed for the random process, ensuring reproducibility of the split (value 6 here can be changed to use a different seed).

Once done, X_train and Y_train will hold the data used for training the model, while X_test and Y_test will hold data used for evaluating the model's performance on unseen data. Splitting the dataset into training and testing sets helps in assessing the accuracy and performance of the machine learning model on new data.

3.6.4 Training an SVM Model

```
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)
model.fit(X_train, Y_train)
```



The screenshot shows a Jupyter Notebook cell with a dropdown menu open. The dropdown menu is titled 'SVC' and shows the constructor 'SVC(kernel='linear', probability=True)'. The cell content is the same as the code block above.

Image 26. Uses SVC from sklearn.svm to create a Support Vector Machine model

from sklearn.svm import SVC: Import SVC class from sklearn.svm module. SVC is used to build an SVM model for the classification problem.

model = SVC(kernel='linear', probability=True): Creates an SVC object

kernel='linear': Select the kernel as 'linear', which means using a linear decision boundary.

probability=True: Enable probability estimation feature. Allows the model to predict probabilities for each class, not just the class with the highest probability.

model.fit(X_train, Y_train): Train the SVC model on the training data (X_train, Y_train). The fit method helps the model learn to classify data based on information from X_train and Y_train labels.

3.6.5 Prediction on Training and Testing Data

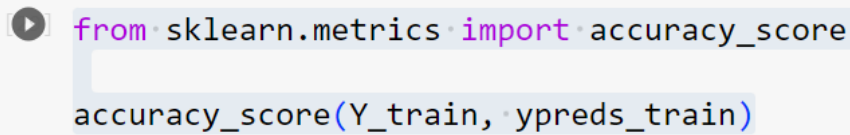
```
[28] ypreds_train = model.predict(X_train)
      ypreds_test = model.predict(X_test)
```

Image 27. Predicts labels for the training data (X_train) and testing data (X_test)

ypreds_train = model.predict(X_train): Using the trained model (model) to predict labels for the training data (X_train). The predicted results will be stored in the variable ypreds_train.

ypreds_test = model.predict(X_test): Employing the trained model (model) to predict labels for the test data (X_test). The predicted results will be stored in the variable ypreds_test.

3.6.6 Calculating Accuracy



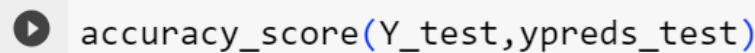
```
from sklearn.metrics import accuracy_score  
  
accuracy_score(Y_train, ypreds_train)
```

1.0

Image 28. Calculate the accuracy of the classification model based on the training set

accuracy_score(Y_train, ypreds_train): Computes the accuracy of the model's predictions on the training dataset by comparing the true labels (Y_train) against the predicted labels (ypreds_train).

The result from the accuracy_score function is 1.0 represents perfect accuracy (100%) on the training dataset.




```
accuracy_score(Y_test, ypreds_test)
```

1.0

Image 29. Calculate the accuracy of the classification model based on the test set

accuracy_score(Y_test, ypreds_test): Calculates the accuracy of the model's predictions on the testing dataset by comparing the true labels (Y_test) against the predicted labels (ypreds_test).

The result from the Precision_score function is 1.0 represents perfect accuracy (100%) on the test set.



```
import pickle  
#save the model  
with open('/content/drive/MyDrive/face_recognition_Facenet/svm_model_160x160.pkl', 'wb') as f:  
    pickle.dump(model, f)
```

Image 30. Save the model

Uses pickle to save the trained model into a file with the path /content/drive/MyDrive/face_recognition_Facenet/svm_model_160x160.pkl.

CHAPTER 4: DEPLOYING REAL-TIME FACIAL RECOGNITION MODEL.

4.1 Initializing Libraries and Models

haarcascade: This is a CascadeClassifier object in OpenCV initialized to use the cascade classifier model for detecting faces within images. The file "**haarcascade_frontalface_default.xml**" contains essential features required for detecting faces in images.

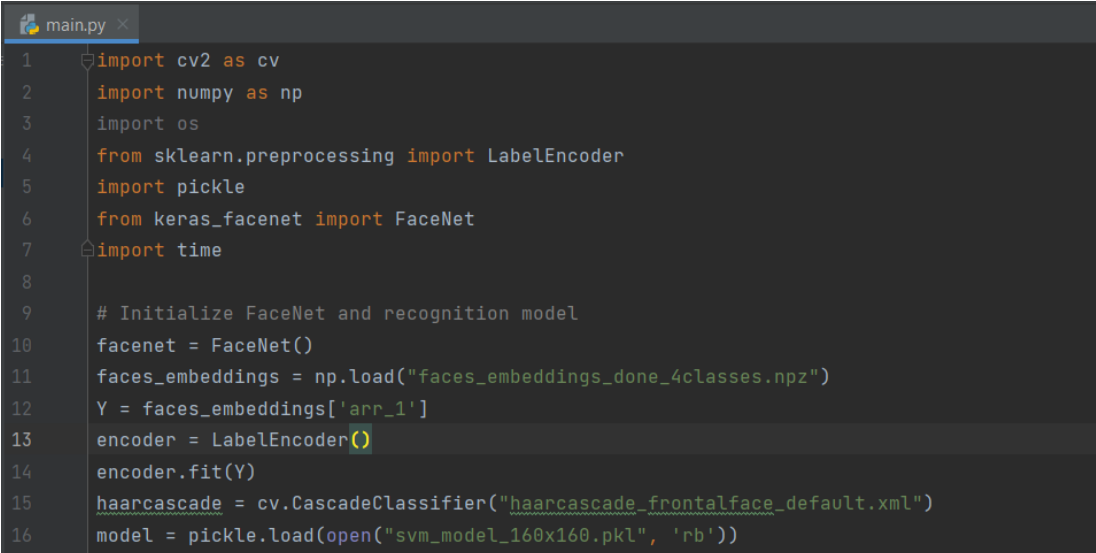
- haarcascade_frontalface_default.xml is a standard file available in the OpenCV library for use in face recognition.

model: It's a machine learning model (SVM - Support Vector Machine) that has been trained and saved as the "**svm_model_160x160.pkl**" file. This model is used for facial recognition after the faces have been embedded into feature vectors.

- svm_model_160x160.pkl is downloaded after successfully training the Facenet model in chapter 3.

faces_embeddings: This file stores the embeddings of pre-processed facial images along with corresponding label information. The file "**faces_embeddings_done_4classes.npz**" contains embeddings of faces along with details about their respective labels.

- Faces_embeddings_done_4classes.npz loads after extracting facial features of the Facenet model in chapter 3.



```
1 import cv2 as cv
2 import numpy as np
3 import os
4 from sklearn.preprocessing import LabelEncoder
5 import pickle
6 from keras_facenet import FaceNet
7 import time
8
9 # Initialize FaceNet and recognition model
10 facenet = FaceNet()
11 faces_embeddings = np.load("faces_embeddings_done_4classes.npz")
12 Y = faces_embeddings['arr_1']
13 encoder = LabelEncoder()
14 encoder.fit(Y)
15 haarcascade = cv.CascadeClassifier("haarcascade_frontalface_default.xml")
16 model = pickle.load(open("svm_model_160x160.pkl", 'rb'))
```

Image 31. Initializing Libraries and Models

It utilizes OpenCV (cv2) for handling video and images.

NumPy (numpy) is used for array and matrix manipulation

The pre-trained model from keras_facenet library is imported.

Data related to facial embeddings, label encoding, cascade classifier, and SVM model are loaded from previously saved files.

4.2 Face Processing and Recognition

```
18 cap = cv.VideoCapture(0)
19
20 # Time start
21 start_time = time.time()
22 frames_processed = 0
23 detection_speed = 0
24
25 while cap.isOpened():
26     _, frame = cap.read()
27     rgb_img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
28     gray_img = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
29
30     # Measure processing time for each frame
31     processing_start = time.time()
32
33     faces = haarcascade.detectMultiScale(gray_img, scaleFactor: 1.3, minNeighbors: 5)
34     for x, y, w, h in faces:
35         img = rgb_img[y:y + h, x:x + w]
36         img = cv.resize(img, dsize: (160, 160))
37         img = np.expand_dims(img, axis=0)
38         ypred = facenet.embeddings(img)
39         face_probabilities = model.predict_proba(ypred)
```

Image 32. Face Processing

Webcam access (cv.VideoCapture(0)) retrieves image data from the camera.

For each frame:

- Conversion of color space from BGR to RGB and grayscale for face detection.
- Use of haarcascade.detectMultiScale() to detect faces in the frame.

```

41         # Determine the class with the highest probability
42         highest_probability_class = np.argmax(face_probabilities)
43         highest_probability = face_probabilities[0, highest_probability_class]
44
45         # Define a threshold to determine "unknown" faces
46         threshold = 0.7
47
48         if highest_probability >= threshold:
49             face_name = highest_probability_class
50             final_name = encoder.inverse_transform(face_name.flatten())[0]
51
52             accuracy = highest_probability # Get the degree value of precision
53         else:
54             final_name = "unknown"
55             accuracy = highest_probability
56
57         # Draw bounding box and label
58         cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 5)
59         # Label accuracy and FPS(Frames Per Second)
60         label_text = f"{final_name} ({accuracy:.2f}) {detection_speed:.2f} FPS"
61         cv.putText(frame, label_text, org=(x, y - 10), cv.FONT_HERSHEY_SIMPLEX,
62                   fontScale=0.6, color=(0, 255, 255), thickness=3, cv.LINE_AA)

```

Image 33. Recognition

For each frame:

- Use of `haarcascade.detectMultiScale()` to detect faces in the frame.
- Calculation of processing time per frame.
- Prediction of detected faces using the pre-trained facial recognition model.
- Drawing bounding boxes around faces and displaying labels containing the recognized person's name and the prediction accuracy on the frame.

4.3 Calculating Face Detection Speed

```

64         # Measure processing time and calculate speed
65         processing_end = time.time()
66         frames_processed += 1
67         elapsed_time = processing_end - start_time
68         if elapsed_time > 1: # Calculate every second
69             detection_speed = frames_processed / elapsed_time
70             frames_processed = 0
71             start_time = time.time()

```

Image 34. Face Detection Speed

Measurement of the face detection speed within a fixed period (1 second) by counting the number of processed frames.

Displaying the face detection speed at the top of the frame.

4.4 Displaying the Video stream and Exiting the Program

```
73     cv.imshow( winname: "Face Recognition", frame)
74
75     key = cv.waitKey(1)
76     if key == ord('q') or key == 27:
77         break
78
79     cap.release()
80     cv.destroyAllWindows()
```

Image 35. Displaying the video stream

Displaying the face detection speed at the top of the frame.

The program terminates when 'q' or the Esc key is pressed.

4.5 Result

Face recognition under normal conditions.

- Not expressive

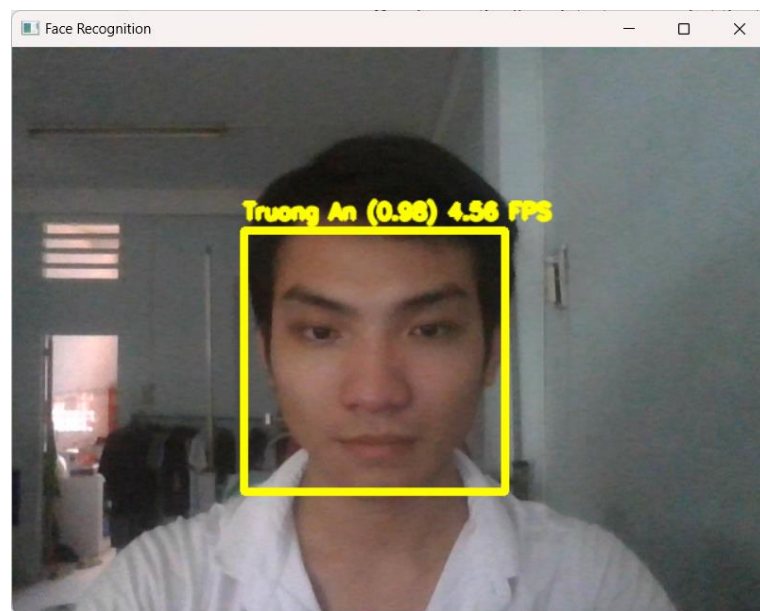


Image 36. Face recognition under normal conditions - Not expressive

- Smiling

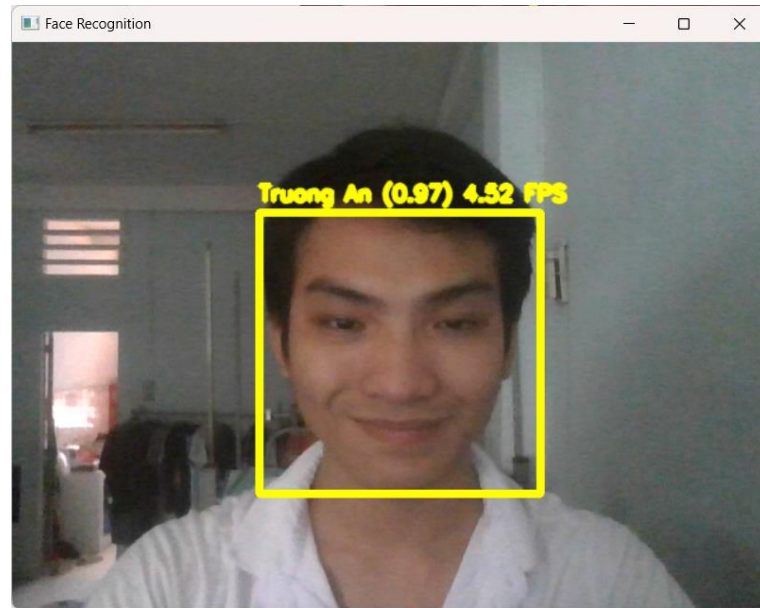


Image 37. Face recognition under normal conditions – Smiling

- Uncomfortable

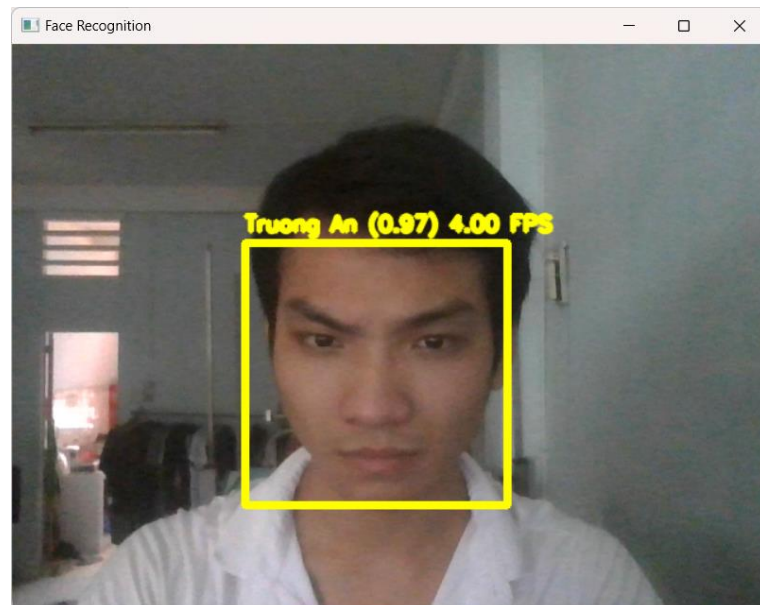


Image 38. Face recognition under normal conditions – Uncomfortable

- Wear glasses

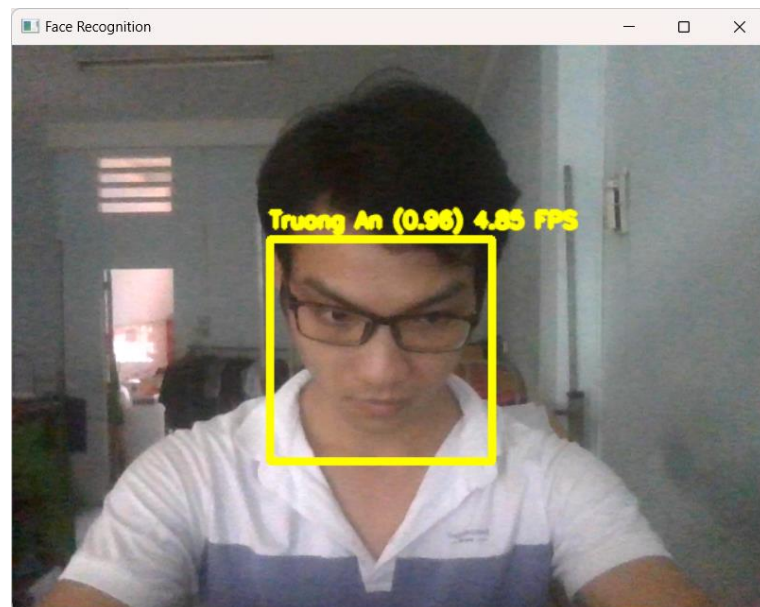


Image 39. Face recognition under normal conditions – Wear glasses

- Left-leaning angle

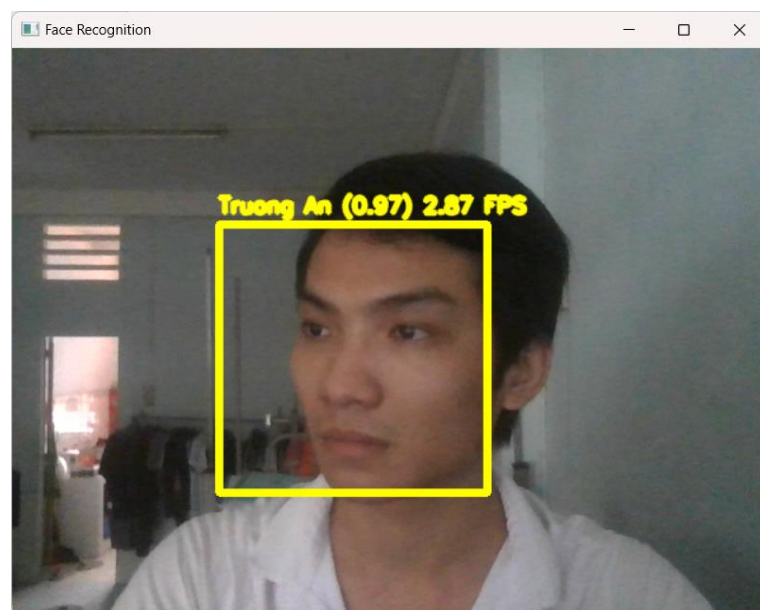


Image 40. Face recognition under normal conditions – Left-leaning angle

- Right-leaning angle

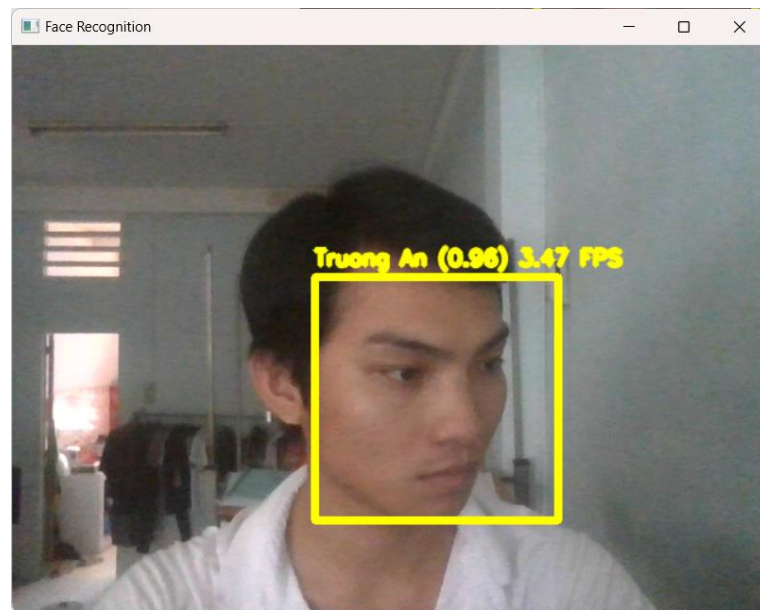


Image 41. Face recognition under normal conditions – Right-leaning angle

- Upward angle

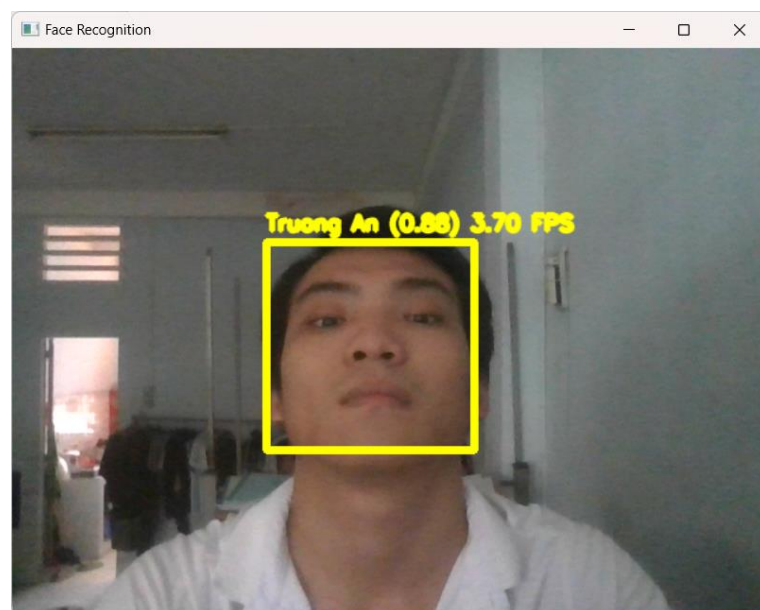


Image 42. Face recognition under normal conditions - Upward angle

- Downward angle

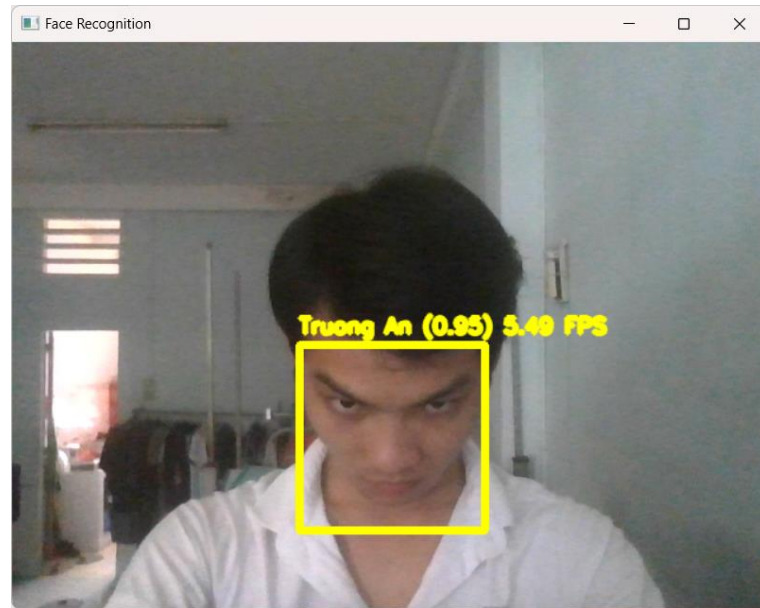


Image 43. Face recognition under normal conditions - Downward angle

Face recognition in low light conditions.

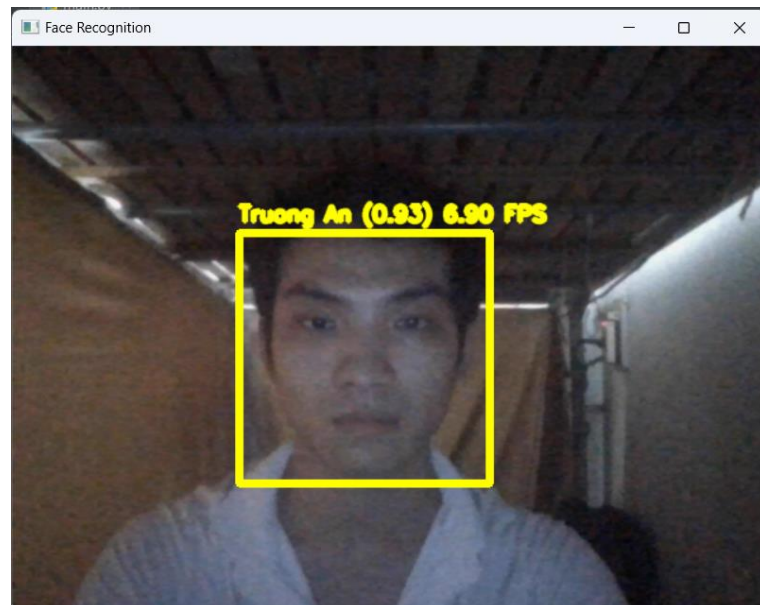


Image 44. Face recognition in low light conditions.

Face recognition in strong light conditions.

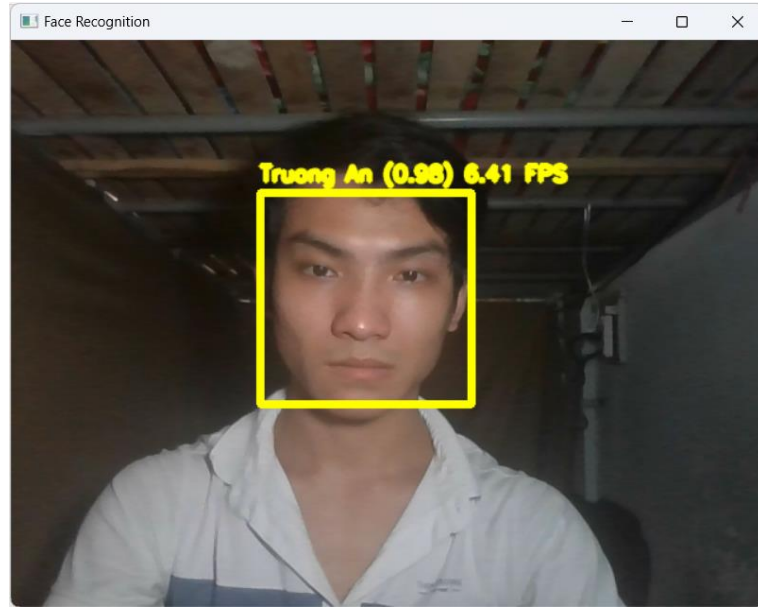


Image 45. Face recognition in strong light conditions.

Condition	Not expressive	Smiling	Uncomfortable	Wear glasses	Left-leaning angle
Accuracy	0.98	0.97	0.97	0.96	0.97
FPS	4.56	4.52	4.00	4.85	2.87

Table 1. All information about recognition accuracy and FPS part 1.

Condition	Right-leaning angle	Upward angle	Downward angle	Low light	Strong light
Accuracy	0.96	0.88	0.95	0.93	0.98
FPS	3.47	3.70	5.49	6.90	6.41

Table 2. All information about recognition accuracy and FPS part 2.

- Facc

CHAPTER 5: TESTING ON TEST SET, COMPARISON, EVALUATION

5.1 Testing on Test set

```
import os
import cv2 as cv
import numpy as np
import time
import warnings
from sklearn.exceptions import DataConversionWarning

warnings.filterwarnings("ignore", category=DataConversionWarning)

detector = MTCNN()

test_folder = "/content/drive/MyDrive/face_recognition_Facenet/test"
predictions = []

for filename in os.listdir(test_folder):
    start_time = time.time()
    test_image = cv.imread(os.path.join(test_folder, filename))
    test_image = cv.cvtColor(test_image, cv.COLOR_BGR2RGB)
    x, y, w, h = detector.detect_faces(test_image)[0]['box']

    face_image = test_image[y:y+h, x:x+w]
    face_image = cv.resize(face_image, (160, 160))
    test_embedding = get_embedding(face_image)

    # Utilize feature vectors for prediction
    ypred = model.predict([test_embedding])

    predicted_name = encoder.inverse_transform([ypred])[0]
    accuracy = model.predict_proba([test_embedding]).max() * 100

    end_time = time.time()
    process_time = end_time - start_time

    predictions.append((filename, predicted_name, accuracy, process_time))
```

Image 46. Recognize faces and predict the identity of the Test set based on a pre-trained model.

Read the image, convert it to RGB format, and use MTCNN to detect faces within the image.

Crop the facial image and resize it to 160x160 to embed it into the model.

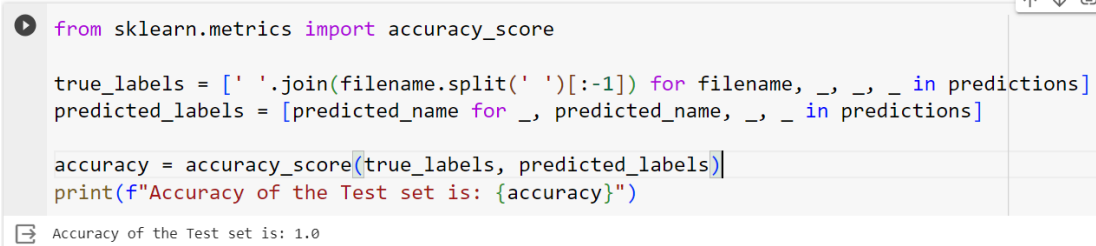
Compute the embedding (feature vector) of the cropped facial image.

Utilize the model to predict the identity of the face from the computed embedding.

Reverse the encoding process to obtain the predicted name.

Calculate the accuracy of the prediction (based on the highest predicted probability).

Record the processing time and save prediction information (filename, predicted name, accuracy, processing time) in the predictions list.



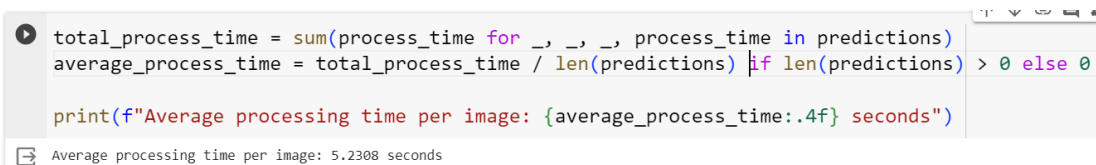
```
from sklearn.metrics import accuracy_score

true_labels = [' '.join(filename.split(' ')[:-1]) for filename, _, _ in predictions]
predicted_labels = [predicted_name for _, predicted_name, _ in predictions]

accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Accuracy of the Test set is: {accuracy}")
```

Accuracy of the Test set is: 1.0

Image 47. The accuracy of the prediction model on the Test set



```
total_process_time = sum(process_time for _, _, _, process_time in predictions)
average_process_time = total_process_time / len(predictions) if len(predictions) > 0 else 0

print(f"Average processing time per image: {average_process_time:.4f} seconds")
```

Average processing time per image: 5.2308 seconds

Image 48. Average processing time per image in Test Set.

Import the `accuracy_score` function from the `scikit-learn` library.

Generate the `true_labels` list, containing the actual labels of the test images. These labels are extracted from the image filenames stored in the `predictions` variable.

Create the `predicted_labels` list, holding the predicted labels from the model for the test images.

Utilize the `accuracy_score` function from `scikit-learn` to compute the accuracy between the true and predicted labels on the test dataset.

The calculated accuracy is stored in the `accuracy` variable.

The resulting accuracy score is displayed on the screen.

```

from prettytable import PrettyTable
table1 = PrettyTable()
table2 = PrettyTable()

# Define field names for both tables
field_names = ["Image", "Predicted", "Accuracy", "Processing Speed", "Result"]

# Add field names to tables
table1.field_names = field_names
table2.field_names = field_names

for filename, predicted_name, accuracy, process_time in predictions:
    true_label = ' '.join(filename.split(' ')[:-1])

    # Define result based on comparison of true_label and predicted_name
    result = "Yes" if true_label == predicted_name else "No"

    # Prepare data for tables based on result
    table_data = [filename, predicted_name, f"{accuracy:.2f}%",
                  f"{process_time:.4f} sec", result]

    if result == "Yes":
        table1.add_row(table_data)
    else:
        table2.add_row(table_data)

# Set title for Table 1
table1.title = "List of correctly predicted images"
if len(table1._rows) > 0:
    print(table1)
else:
    print("There are no correct predictions")
print('\n')

table2.title = "List of incorrectly predicted images"
if len(table2._rows) > 0:
    print(table2)
else:
    print("There are no wrong predictions")

```

Image 49. Print out the prediction information table for each image

Creates two tables: one displaying correctly predicted images and the other displaying incorrectly predicted images.

It populates these tables by iterating through the predictions list, categorizing the results, and adding relevant data to the appropriate table.

Finally, it displays these tables with their respective titles, indicating whether there are any correct or incorrect predictions.

List of correctly predicted images					
Image	Predicted	Accuracy	Processing Speed	Result	
Bich Huyen 1.jpg	Bich Huyen	93.50%	5.7524 sec	Yes	
Bich Huyen 3.jpg	Bich Huyen	79.87%	5.1949 sec	Yes	
Bich Huyen 5.jpg	Bich Huyen	82.51%	6.3405 sec	Yes	
Thi Dong 7.jpg	Thi Dong	81.92%	1.6544 sec	Yes	
Thi Dong 8.jpg	Thi Dong	86.00%	2.8117 sec	Yes	
Truong An 10.jpg	Truong An	98.29%	5.9392 sec	Yes	
Truong An 8.jpg	Truong An	95.15%	4.3095 sec	Yes	
Truong An 6.jpg	Truong An	95.72%	5.0250 sec	Yes	
Hoang Hao 4.jpg	Hoang Hao	87.90%	6.6750 sec	Yes	
Hoang Hao 3.jpg	Hoang Hao	82.20%	5.0867 sec	Yes	
Hoang Hao 6.jpg	Hoang Hao	89.74%	4.7321 sec	Yes	
Hoang Hao 9.jpg	Hoang Hao	81.90%	7.2389 sec	Yes	
Hoang Hao 10.jpg	Hoang Hao	87.36%	4.8305 sec	Yes	
Hoang Hao 7.jpg	Hoang Hao	83.70%	4.6661 sec	Yes	
Hoang Hao 1.jpg	Hoang Hao	84.70%	6.6256 sec	Yes	
Hoang Hao 8.jpg	Hoang Hao	80.76%	4.7442 sec	Yes	
Dang Tam 1.jpg	Dang Tam	87.10%	5.1537 sec	Yes	
Dang Tam 2.jpg	Dang Tam	88.09%	8.6478 sec	Yes	
Dang Tam 3.jpg	Dang Tam	87.20%	4.9950 sec	Yes	
Dang Tam 4.jpg	Dang Tam	79.95%	4.7892 sec	Yes	
Dang Tam 5.jpg	Dang Tam	78.35%	7.8316 sec	Yes	
Dang Tam 6.jpg	Dang Tam	87.14%	5.1411 sec	Yes	
Dang Tam 7.jpg	Dang Tam	83.61%	5.5192 sec	Yes	
Dang Tam 8.jpg	Dang Tam	80.49%	6.4625 sec	Yes	
Dang Tam 9.jpg	Dang Tam	85.50%	5.0030 sec	Yes	
Dang Tam 10.jpg	Dang Tam	82.94%	5.6955 sec	Yes	
Quoc Vinh 1.jpg	Quoc Vinh	84.55%	7.2313 sec	Yes	
Quoc Vinh 2.jpg	Quoc Vinh	87.05%	4.7614 sec	Yes	
Quoc Vinh 3.jpg	Quoc Vinh	89.59%	4.7837 sec	Yes	
Quoc Vinh 4.jpg	Quoc Vinh	87.73%	7.0523 sec	Yes	
Quoc Vinh 5.jpg	Quoc Vinh	89.80%	4.8724 sec	Yes	
Quoc Vinh 6.jpg	Quoc Vinh	85.65%	5.2551 sec	Yes	

Image 50. List of correctly predicted images – part 1

Quoc Vinh 7.jpg	Quoc Vinh	84.28%	7.3231 sec	Yes
Quoc Vinh 8.jpg	Quoc Vinh	85.79%	4.4926 sec	Yes
Quoc Vinh 9.jpg	Quoc Vinh	76.28%	5.8562 sec	Yes
Quoc Vinh 10.jpg	Quoc Vinh	90.34%	6.4454 sec	Yes
Bich Huyen 2.jpg	Bich Huyen	89.71%	3.3006 sec	Yes
Bich Huyen 4.jpg	Bich Huyen	93.08%	3.6108 sec	Yes
Bich Huyen 6.jpg	Bich Huyen	92.61%	10.4556 sec	Yes
Bich Huyen 7.jpg	Bich Huyen	88.06%	3.6331 sec	Yes
Bich Huyen 8.jpg	Bich Huyen	76.76%	8.2701 sec	Yes
Bich Huyen 10.jpg	Bich Huyen	84.90%	7.1298 sec	Yes
Thi Dong 1.jpg	Thi Dong	75.96%	2.1037 sec	Yes
Thi Dong 2.jpg	Thi Dong	83.29%	2.4423 sec	Yes
Thi Dong 3.jpg	Thi Dong	77.20%	2.7539 sec	Yes
Thi Dong 4.jpg	Thi Dong	76.03%	2.4888 sec	Yes
Thi Dong 5.jpg	Thi Dong	73.78%	2.9273 sec	Yes
Thi Dong 6.jpg	Thi Dong	69.40%	2.3612 sec	Yes
Truong An 1.jpg	Truong An	96.46%	4.7901 sec	Yes
Truong An 2.jpg	Truong An	95.25%	6.1651 sec	Yes
Truong An 5.jpg	Truong An	97.05%	5.2570 sec	Yes
Truong An 4.jpg	Truong An	96.73%	4.5535 sec	Yes
Thi Dong 9.jpg	Thi Dong	92.60%	1.9068 sec	Yes
Thi Dong 10.jpg	Thi Dong	89.75%	2.2624 sec	Yes
Truong An 7.jpg	Truong An	96.46%	6.5738 sec	Yes
Truong An 3.jpg	Truong An	96.36%	4.4069 sec	Yes
Truong An 9.jpg	Truong An	91.95%	5.5728 sec	Yes
Hoang Hao 2.jpg	Hoang Hao	79.94%	7.5196 sec	Yes
Hoang Hao 5.jpg	Hoang Hao	82.98%	4.8239 sec	Yes
Bich Huyen 9.jpg	Bich Huyen	82.51%	8.2712 sec	Yes
Truong An 11.jpg	Truong An	95.77%	5.3162 sec	Yes
Truong An 12.jpg	Truong An	95.06%	4.5453 sec	Yes
Truong An 13.jpg	Truong An	96.09%	5.7564 sec	Yes
Truong An 14.jpg	Truong An	94.82%	6.3076 sec	Yes
Truong An 15.jpg	Truong An	94.52%	4.9164 sec	Yes
Truong An 16.jpg	Truong An	97.80%	5.5262 sec	Yes

Image 51. List of correctly predicted images – part 2

Truong An 17.jpg	Truong An	93.99%	10.5932 sec	Yes
Truong An 18.jpg	Truong An	89.06%	10.0696 sec	Yes
Truong An 19.jpg	Truong An	94.60%	5.1042 sec	Yes
Truong An 20.jpg	Truong An	96.38%	4.3001 sec	Yes
Truong An 21.jpg	Truong An	93.54%	5.4549 sec	Yes
Truong An 22.jpg	Truong An	95.25%	6.0285 sec	Yes
Truong An 23.jpg	Truong An	95.26%	4.6766 sec	Yes
Truong An 24.jpg	Truong An	93.70%	5.2065 sec	Yes
Truong An 25.jpg	Truong An	95.15%	6.5655 sec	Yes
Truong An 26.jpg	Truong An	94.50%	4.4223 sec	Yes
Truong An 27.jpg	Truong An	96.28%	4.9799 sec	Yes
Truong An 28.jpg	Truong An	97.22%	7.6139 sec	Yes
Truong An 29.jpg	Truong An	95.98%	5.0735 sec	Yes
Truong An 30.jpg	Truong An	97.41%	4.5892 sec	Yes
Truong An 31.jpg	Truong An	98.21%	7.0874 sec	Yes
Truong An 32.jpg	Truong An	80.50%	4.7620 sec	Yes
Truong An 33.jpg	Truong An	98.59%	4.6930 sec	Yes
Truong An 34.jpg	Truong An	96.65%	8.3657 sec	Yes
Truong An 35.jpg	Truong An	97.22%	4.6060 sec	Yes
Truong An 36.jpg	Truong An	96.19%	6.1969 sec	Yes
Truong An 37.jpg	Truong An	96.96%	6.7890 sec	Yes
Truong An 38.jpg	Truong An	97.01%	5.0080 sec	Yes
Truong An 39.jpg	Truong An	96.23%	5.8784 sec	Yes
Truong An 40.jpg	Truong An	95.79%	6.8874 sec	Yes
Truong An 41.jpg	Truong An	95.51%	5.9802 sec	Yes
Truong An 42.jpg	Truong An	95.28%	12.4791 sec	Yes
Truong An 43.jpg	Truong An	95.75%	5.3459 sec	Yes
Truong An 44.jpg	Truong An	96.21%	6.0983 sec	Yes
Truong An 46.jpg	Truong An	94.40%	6.1785 sec	Yes
Truong An 48.jpg	Truong An	90.05%	4.4378 sec	Yes
Truong An 47.jpg	Truong An	82.44%	5.7114 sec	Yes
Truong An 45.jpg	Truong An	96.74%	5.5856 sec	Yes
Truong An 49.jpg	Truong An	90.72%	4.8882 sec	Yes
Truong An 50.jpg	Truong An	95.73%	4.7218 sec	Yes

Image 52. List of correctly predicted images – part 3

There are no wrong predictions

Image 53. List of incorrectly predicted images

List of incorrectly predicted images.


```
total_accuracy = 0
num_correct_predictions = 0

for row in table1._rows:
    accuracy_str = row[field_names.index("Accuracy")]
    accuracy = float(accuracy_str[:-1])
    total_accuracy += accuracy
    num_correct_predictions += 1

average_accuracy = total_accuracy / num_correct_predictions if num_correct_predictions > 0 else 0
print(f"Average accuracy of correctly predicted images: {average_accuracy:.2f}%")
```

Average accuracy of correctly predicted images: 89.45%

Image 54. Average accuracy of correctly predicted images.

5.2 Comparison with Resnet model

5.2.1 Deploy the Resnet model

ResNet (Residual Network):

ResNet is a deep neural network architecture known for its depth. It introduced the concept of residual blocks, addressing the vanishing gradient problem by using skip connections (or shortcuts) to jump over layers. This helps in training very deep networks effectively by enabling the flow of gradients throughout the network.

```
import cv2 as cv
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from torchvision import transforms
from torchvision.models import resnet34
import torch
from mtcnn.mtcnn import MTCNN
detector = MTCNN()
```

Image 55. Import library for Resnet model

Uses the MTCNN (Multi-task Cascaded Convolutional Networks) model to detect faces within images. Subsequently, it employs the ResNet34 model to train an SVM (Support Vector Machine) model for facial recognition.

```

▶ # Load the ResNet model
resnet_model = resnet34(pretrained=True)
resnet_model.eval()

/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The path
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Argument
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/tor
100%|██████████| 83.3M/83.3M [00:00<00:00, 109MB/s]
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

```

Image 56. Load the ResNet model

```

▶ from mtcnn.mtcnn import MTCNN
class FACELOADING:
    def __init__(self, directory):
        self.directory = directory
        self.target_size = (224, 224)
        self.X = []
        self.Y = []
        self.detector = MTCNN()
        self.extracted_info = []
        self.unextracted_images = []

```

Image 57. Initialize the FACELOADING object for the Resnet model

Different from the Facenet model, the Resnet model uses images with dimensions of 224 x 224 instead of 160 x 160 like the Facenet model.

```

▶ # Load images and labels from folder
faceloading = FACELOADING("/content/drive/MyDrive/face_recognition_Resnet/dataset")

X, Y = faceloading.load_classes()
...
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
2/2 [=====] - 0s 11ms/step

```

Image 58. Load images and labels from dataset of the Resnet model

```

# Image preprocessing and embedding into feature space using ResNet model
EMBEDDED_X = []
X_np = np.array(X)

def get_embedding_resnet(face_img, resnet_model):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    # Convert PIL Image to PyTorch tensor
    face_tensor = transform(face_img).unsqueeze(0)

    with torch.no_grad():
        embedding = resnet_model(face_tensor)

    return embedding.flatten().numpy()

for img in X_np:
    img_pil = Image.fromarray((img * 255).astype(np.uint8))
    embedding = get_embedding_resnet(img_pil, resnet_model)
    EMBEDDED_X.append(embedding)

EMBEDDED_X = np.asarray(EMBEDDED_X)

```

Image 59. Image preprocessing and embedding into feature space using ResNet model

```

# Evaluate the model on the training and test sets
ypreds_train = svm_model.predict(X_train)
ypreds_test = svm_model.predict(X_test)

[12] train_accuracy = accuracy_score(Y_train, ypreds_train)
     test_accuracy = accuracy_score(Y_test, ypreds_test)

[13] print(f"Training Accuracy: {train_accuracy}")
     print(f"Testing Accuracy: {test_accuracy}")

Training Accuracy: 1.0
Testing Accuracy: 0.9466666666666667

```

Image 60. Evaluate the model on the training and test sets of the Resnet model.

```

from sklearn.metrics import accuracy_score

true_labels = [' '.join(filename.split(' ')[:-1]) for filename, _, _ in predictions]
predicted_labels = [predicted_name for _, predicted_name, _ in predictions]

accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Accuracy of the Test set is: {accuracy}")

Accuracy of the Test set is: 78.00%

```

Image 61. The accuracy of the prediction Resnet model on the Test set

```
[16] total_process_time = sum(process_time for _, _, _, process_time in predictions)
    average_process_time = total_process_time / len(predictions) if len(predictions) > 0 else 0

    print(f"Average processing time per image: {average_process_time:.4f} seconds")
```

Average processing time per image: 4.7767 seconds

Image 62. Average processing time per image in Test Set of the Resnet model.

List of correctly predicted images				
Image	Predicted	Accuracy	Processing Speed	Result
Bich Huyen 1.jpg	Bich Huyen	99.42%	4.1156 sec	Yes
Hoang Hao 5.jpg	Hoang Hao	75.85%	8.3286 sec	Yes
Hoang Hao 10.jpg	Hoang Hao	76.32%	4.4345 sec	Yes
Hoang Hao 4.jpg	Hoang Hao	78.75%	6.0142 sec	Yes
Hoang Hao 2.jpg	Hoang Hao	47.28%	4.2694 sec	Yes
Hoang Hao 3.jpg	Hoang Hao	83.56%	4.0691 sec	Yes
Dang Tam 8.jpg	Dang Tam	48.26%	5.1640 sec	Yes
Dang Tam 9.jpg	Dang Tam	62.16%	5.1372 sec	Yes
Hoang Hao 1.jpg	Hoang Hao	95.34%	5.0265 sec	Yes
Dang Tam 10.jpg	Dang Tam	58.85%	4.8057 sec	Yes
Dang Tam 1.jpg	Dang Tam	35.31%	6.0982 sec	Yes
Bich Huyen 7.jpg	Bich Huyen	90.81%	3.4911 sec	Yes
Bich Huyen 10.jpg	Bich Huyen	75.27%	5.5555 sec	Yes
Bich Huyen 4.jpg	Bich Huyen	68.86%	3.9674 sec	Yes
Bich Huyen 2.jpg	Bich Huyen	46.65%	3.0095 sec	Yes
Thi Dong 5.jpg	Thi Dong	51.77%	1.4092 sec	Yes
Thi Dong 10.jpg	Thi Dong	70.26%	1.7415 sec	Yes
Thi Dong 2.jpg	Thi Dong	42.28%	3.0685 sec	Yes
Quoc Vinh 9.jpg	Quoc Vinh	93.44%	5.1977 sec	Yes
Quoc Vinh 7.jpg	Quoc Vinh	94.84%	3.9287 sec	Yes
Quoc Vinh 8.jpg	Quoc Vinh	88.28%	4.6542 sec	Yes
Quoc Vinh 6.jpg	Quoc Vinh	82.62%	5.5526 sec	Yes
Quoc Vinh 4.jpg	Quoc Vinh	65.71%	4.4343 sec	Yes
Quoc Vinh 5.jpg	Quoc Vinh	75.22%	4.4101 sec	Yes
Quoc Vinh 10.jpg	Quoc Vinh	93.82%	6.4067 sec	Yes
Quoc Vinh 3.jpg	Quoc Vinh	89.03%	4.7142 sec	Yes
Quoc Vinh 2.jpg	Quoc Vinh	69.33%	3.7685 sec	Yes
Quoc Vinh 1.jpg	Quoc Vinh	46.84%	4.4540 sec	Yes
Hoang Hao 6.jpg	Hoang Hao	93.44%	3.8166 sec	Yes
Hoang Hao 8.jpg	Hoang Hao	87.80%	4.2315 sec	Yes
Hoang Hao 7.jpg	Hoang Hao	94.28%	5.5398 sec	Yes
Hoang Hao 9.jpg	Hoang Hao	42.44%	5.8489 sec	Yes
Truong An 24.jpg	Truong An	93.53%	4.4213 sec	Yes
Truong An 27.jpg	Truong An	98.71%	5.3760 sec	Yes
Truong An 25.jpg	Truong An	70.63%	4.3716 sec	Yes
Truong An 21.jpg	Truong An	93.81%	4.8725 sec	Yes
Truong An 23.jpg	Truong An	90.79%	6.1990 sec	Yes
Truong An 22.jpg	Truong An	96.50%	4.9537 sec	Yes
Truong An 2.jpg	Truong An	96.50%	5.0423 sec	Yes
Truong An 17.jpg	Truong An	72.64%	6.1950 sec	Yes
Truong An 18.jpg	Truong An	85.51%	4.5362 sec	Yes
Truong An 16.jpg	Truong An	86.61%	4.5991 sec	Yes
Truong An 20.jpg	Truong An	84.17%	5.6990 sec	Yes
Truong An 19.jpg	Truong An	88.48%	5.5764 sec	Yes
Truong An 15.jpg	Truong An	92.33%	4.7502 sec	Yes
Truong An 14.jpg	Truong An	96.42%	4.7418 sec	Yes
Truong An 11.jpg	Truong An	70.96%	5.6669 sec	Yes
Truong An 12.jpg	Truong An	80.23%	4.4832 sec	Yes
Truong An 13.jpg	Truong An	95.94%	3.9041 sec	Yes
Thi Dong 7.jpg	Thi Dong	92.59%	1.5637 sec	Yes
Truong An 10.jpg	Truong An	98.05%	5.6421 sec	Yes
Thi Dong 9.jpg	Thi Dong	42.36%	2.3921 sec	Yes
Thi Dong 8.jpg	Thi Dong	77.61%	1.9562 sec	Yes

Image 63. List of correctly predicted images of the Resnet model.

List of incorrectly predicted images					
Image	Predicted	Accuracy	Processing Speed	Result	
Bich Huyen 9.jpg	Truong An	50.55%	6.0405 sec	No	
Dang Tam 7.jpg	Hoang Hao	47.96%	4.2254 sec	No	
Bich Huyen 8.jpg	Thi Dong	52.31%	7.7251 sec	No	
Dang Tam 6.jpg	Hoang Hao	50.70%	5.7602 sec	No	
Bich Huyen 6.jpg	Truong An	45.01%	11.0565 sec	No	
Dang Tam 3.jpg	Hoang Hao	45.54%	4.5122 sec	No	
Dang Tam 2.jpg	Hoang Hao	53.73%	4.5175 sec	No	
Bich Huyen 5.jpg	Truong An	50.55%	5.4502 sec	No	
Dang Tam 5.jpg	Truong An	97.30%	5.7863 sec	No	
Bich Huyen 3.jpg	Truong An	76.74%	4.8316 sec	No	
Thi Dong 6.jpg	Truong An	89.43%	1.8898 sec	No	
Thi Dong 3.jpg	Truong An	46.67%	1.4213 sec	No	
Thi Dong 4.jpg	Truong An	37.41%	1.3868 sec	No	
Thi Dong 1.jpg	Bich Huyen	86.04%	1.6481 sec	No	
Dang Tam 4.jpg	Hoang Hao	49.57%	6.6782 sec	No	
Truong An 26.jpg	Thi Dong	45.74%	5.5069 sec	No	
Truong An 1.jpg	Hoang Hao	49.45%	4.8055 sec	No	
Truong An 48.jpg	Quoc Vinh	56.59%	4.0171 sec	No	
Truong An 41.jpg	Bich Huyen	44.28%	4.1826 sec	No	
Truong An 38.jpg	Hoang Hao	41.11%	4.8307 sec	No	
Truong An 32.jpg	Dang Tam	60.92%	4.1264 sec	No	
Truong An 7.jpg	Hoang Hao	49.45%	7.0894 sec	No	

Image 64. List of incorrectly predicted images of the Resnet model

```

total_accuracy = 0
num_correct_predictions = 0

for row in table1._rows:
    accuracy_str = row[field_names.index("Accuracy")]
    accuracy = float(accuracy_str[:-1])
    total_accuracy += accuracy
    num_correct_predictions += 1

average_accuracy = total_accuracy / num_correct_predictions if num_correct_predictions > 0 else 0
print(f"Average accuracy of correctly predicted images of the Resnet model: {average_accuracy:.2f}%")

```

Average accuracy of correctly predicted images of the Resnet model: 80.50%

Image 65. Average accuracy of correctly predicted images of the Resnet model

5.2.2 Compare the Facenet model with the Resnet model

Evaluate the model on the training and test set

	Facenet model	Resnet model
Training accuracy	1.0	1.0
Test accuracy	1.0	0.9667

Table 3. Evaluate the model on the training and test set

The accuracy of the prediction model on the Test set

	Facenet model	Resnet model
Accuracy of the prediction	100%	78%

Table 4. The accuracy of the prediction model on the Test set

Average processing time per image in Test set

	Facenet model	Resnet model
Average processing time	5.2308 seconds	4.7767 seconds

Table 5. Average processing time per image in Test set

Average accuracy of correctly predicted images

	Facenet model	Resnet model
Average accuracy of correctly predicted images	89.45%	80.50%

Table 6. Average accuracy of correctly predicted images

5.3 Evaluation

Architecture: FaceNet is specifically designed for face recognition tasks, using a Siamese network architecture with a triplet loss function. On the other hand, ResNet is a general-purpose architecture initially designed for image classification but has been widely used in various computer vision tasks.

Training: FaceNet is trained with a large dataset of face images using a specific loss function designed for face recognition. ResNet, being a versatile architecture, can be trained on various datasets and used for different tasks.

Feature Extraction: FaceNet focuses on generating embeddings specifically tailored for face recognition, while ResNet provides high-level features that are useful for general image understanding tasks.

The evaluation of the models on the training and test sets shows that the FaceNet model performs exceptionally well, achieving a perfect accuracy of 1.0 on both the training and test datasets. However, the ResNet model demonstrates a training accuracy of 1.0 while its accuracy on the test set is 0.9667.

The prediction accuracy on Test set reaches a maximum of 100% in the Facenet model. In the Resnet model, it only reaches 78%.

Average accuracy of correctly predicted images in two different models, the Facenet model achieves 89.45 percent accuracy, and the Resnet model reaches 80.50 percent.

Average processing time per image in Test set in Resnet model is 4.7767 seconds and in Facenet model is 5.2308 seconds.

CHAPTER 6: CONCLUSION

6.1 Summary of Achievements

Facial Recognition Model Development: Successfully implemented a facial recognition system leveraging MTCNN for face detection and Facenet for face embedding. Employed SVM as a classification method to recognize faces based on the embeddings.

Integration and Deployment: Successfully integrated MTCNN, SVM, and Facenet into a cohesive system capable of real-time face recognition.

PyCharm Implementation: Deployed the developed facial recognition system in PyCharm, allowing for real-time face recognition using a camera.

6.2 Limitations of the model

Training data is limited.

6.3 Proposed Future Development Directions

Performance Optimization: Further enhance the system's efficiency and processing speed to handle larger datasets and real-time scenarios more effectively.

Adaptive Model for Various Environments: Improve the system's robustness against changes in lighting conditions, camera angles, and individual facial characteristics to ensure accurate recognition in diverse environments.

Exploration of Advanced Models: Explore and experiment with advanced models or architectures to improve accuracy and generalization of the facial recognition system.

User Interface Enhancement: Develop a more user-friendly and intuitive interface for the PyCharm implementation, enabling easy access and utilization of the facial recognition system.

Privacy and Security Considerations: Address privacy concerns and implement security measures to ensure the ethical use of the facial recognition system.

REFERENCES

- [1] “Introduction to support vector machines SVM”
<https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/>
- [2] “A brief history of facial recognition”
<https://www.nec.co.nz/market-leadership/publications-media/a-brief-history-of-facial-recognition/>
- [3] “Paper about Schroff FaceNet A Unified 2015”
https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Schroff_FaceNet_A_Unified_2015_CVPR_paper.pdf
- [4] “Face Recognition using Local Binary Patterns LBP”
https://www.researchgate.net/publication/331645500_Face_Recognition_using_Local_Binary_Patterns_LBP#:~:text=The%20face%20area%20is%20first,to%20measure%20similarities%20between%20images.
- [5] “Oriented Gradients HOG parameters for Facial Recognition”
https://www.researchgate.net/publication/372288882_Bayesian_Optimization_of_Histogram_of_Oriented_Gradients_HOG_parameters_for_Facial_Recognition
- [6] “Face recognition based on convolutional neural network”
<https://ieeexplore.ieee.org/abstract/document/8248937>
- [7] “MTCNN model used for face detection and landmark extraction”
https://www.researchgate.net/figure/MTCNN-Stage-architecture-of-the-model-used-for-face-detection-and-landmark-extraction_fig3_341148320