

UNIVERSITY OF LONDON

INTERNATIONAL PROGRAMMES

BSc Computer Science (Machine Learning and Artificial Intelligence)



CM3070 PROJECT FINAL PROJECT REPORT

< Facial Recognition Through Siamese Networks >

Author: Hao Teck Tey

Student Number : 200261160

Date of Submission: 14 September 2023

Supervisor : Dr Tarapong Sreenuch

Contents

CHAPTER 1:	INTRODUCTION.....	3
CHAPTER 2:	LITERATURE REVIEW	5
CHAPTER 3:	PROJECT DESIGN.....	10
CHAPTER 4:	IMPLEMENTATION	14
CHAPTER 5:	EVALUATION	17
CHAPTER 6:	CONCLUSION	20
CHAPTER 7:	APPENDICES	22
CHAPTER 8:	REFERENCES.....	28

CHAPTER 1: INTRODUCTION

Project Template 1: Deep Learning on a Public Dataset (CM3015 Machine Learning and Neural Networks)

Project Idea: Enhancing Identity Verification through Facial Recognition for Attendance Tracking

Project Concept

With the advent of the Internet, the issue of identity verification has become increasingly significant, finding applications in diverse fields such as attendance marking. In the past, conventional identification methods primarily revolved around fingerprinting or employing smart cards with embedded tags. However, these approaches have exhibited several inherent drawbacks; for instance, fingerprints can be relatively easily replicated, and smart cards are susceptible to loss or misplacement.

Consequently, a more robust and effective solution for identification has emerged, namely facial recognition. This technology guarantees both precision and authenticity in verifying individuals. Therefore, the central objective of this project is to develop a sophisticated facial identification system tailored specifically for the purpose of accurately recording attendance.

Motivation

The driving force behind the development of this model originates from the aspiration to revolutionize the current attendance marking system. The existing system I engage with utilizes Bluetooth technology, mandating the close proximity of my phone to the designated device to establish a connection. Once this connection is validated, the attendance is duly recorded. However, a notable drawback of this system is its sole reliance on device presence, rather than personal presence. This inherent flaw opens the door to potential misuse, wherein a single device could be exploited to log into multiple accounts and subsequently mark attendance for all, in a consolidated manner. This conspicuous flaw underscores the ineffectiveness of adopting such a system.

Here are some questions that we will look at:

- How does a Siamese network contribute to face recognition?
- Which classification method will be employed to accurately identify each individual?

Deliverables

A Siamese network is a specialized type of neural network designed to discern the similarities between two provided images. This architecture consists of dual inputs, both of which undergo an identical embedding process. When these inputs pertain to the same individual or category, the model actively diminishes the distance separating them. Conversely, when the inputs are unrelated in terms of the person or category, the model strives to amplify the distance between them.

By employing a Siamese network architecture, the proximity between similar images is enhanced, effectively reducing the distance within the embedding space. This characteristic proves invaluable for implementing the K-Nearest Neighbors (K-NN) algorithm as our chosen classification methodology. For every individual image embedded by the Siamese Network, a corresponding similar image is situated in its immediate vicinity. This nearby image becomes its nearest neighbor, a pivotal concept in the K-NN algorithm.

Dataset

The dataset utilized in this study has been sourced from Kaggle and is conveniently accessible through the provided [link](#). Governed by a CC0: Public Domain license issued by Creative Commons Corporation, this license ensures that the dataset is freely available to the public, unencumbered by any copyright restrictions.

This dataset contains images of 17 Hollywood celebrities. Each celebrity has 100 images except Scarlett Johansson has 200 images, making a total of 1800 images. I divided the dataset into three parts: 70% (1260 images) for training, 15% (270 images) for validation, and another 15% (270 images) for testing Siamese model. The entire dataset takes up about 54.17 megabytes of storage space. The dataset is organized neatly with 17 separate folders, one for each celebrity, and all images are in JPG format.

Justification

By leveraging the Siamese model and KNN, a comprehensive facial recognition process has been established. The initial step involves channeling raw input through a Siamese network for embedding purposes. Subsequently, this embedded data traverses through a KNN model to identify its closest neighbors. This collective information culminates in the classification of the individual's identity. Once the person is successfully recognized, their identity can be utilized for the purpose of marking attendance.

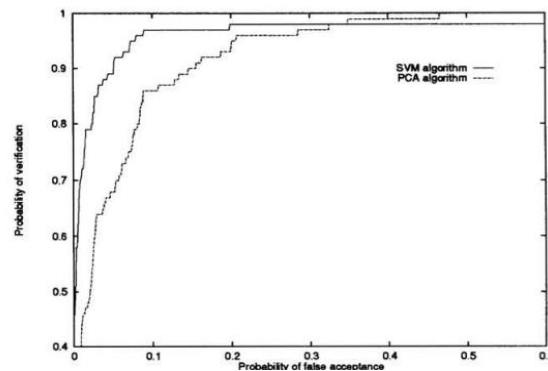
CHAPTER 2: LITERATURE REVIEW

Face Recognition using Support Vector Machine

The document titled "Support Vector Machines Applied to Face Recognition" presents a comprehensive approach to face recognition using Support Vector Machines (SVMs).

The paper introduces a novel SVM-based algorithm for face recognition and demonstrates its efficacy through experimentation. (P. Jonathon Phillips)

The document proposes a new formulation of face recognition as a problem of measuring dissimilarities between facial images. It introduces two classes: within-class differences (dissimilarities within the same individual's facial images) and between-class differences (dissimilarities among different individuals' facial images). By modifying the interpretation of the SVM decision surface, the authors generate a similarity metric between faces that serves as the basis for the SVM-based verification and identification algorithms. In this document, they compare SVM-based algorithm to a PCA-based one using the FERET facial image database. The results clearly show that SVM outperforms PCA, as seen in the ROC curve where SVM consistently outshines PCA.



(Figure 1, ROC Curve comparison of SVM and PCA)

This work motivates the Siamese network project by revealing certain limitations of traditional algorithms. While the SVM-based approach offers improvements in recognition accuracy, it still faces challenges in dealing with varying lighting conditions, pose changes, and limited training samples per individual. The Siamese network project comes into play, as it leverages its inherent ability to learn feature representations that are robust to such variations and requires only pairs of images for training. This document thus highlights the need for approaches like Siamese networks that can generalize well and perform effectively even with a small number of training samples per class.

The presented document supports the Siamese project by suggesting ideas for enhancing face recognition systems. The notion of modeling dissimilarities between images aligns with the Siamese network's approach of learning similarity metrics directly from pairs of images. The Siamese architecture's ability to handle one-shot and few-shot recognition scenarios makes it a suitable candidate for addressing the limitations of the SVM-based method, where there might be only one training sample per person.

The credibility of this source is supported by several factors. Firstly, it is authored by P. Jonathon Phillips, a researcher associated with the National Institute of Standards and Technology (NIST), a reputable institution. Additionally, the paper references previous works, citing contributions from other researchers in the field. The references include peer-reviewed conference proceedings and journals, signifying that the work is situated within the broader academic discourse.

Face Recognition using Eigenfaces

The document, titled "Face Recognition Using Eigenfaces" delves into the topic of face recognition using eigenfaces, a technique based on principal component analysis (PCA). The document outlines the concept of face space, a mathematical representation of facial images, and how eigenfaces can be used to project and recognize faces in this space. The study focuses on the use of eigenfaces for face recognition, discussing experiments conducted to assess the viability of the approach, including variations in lighting, scale, and orientation. (Matthew A. Turk and Alex P. Pentland, 1991)

The eigenface approach introduced in the document lays the foundation for discussing its relevance to a face recognition project involving Siamese networks. This literature contributes to a Siamese network project by highlighting the importance of feature extraction and dimensionality reduction in face recognition. The concept of eigenfaces aligns with the idea of learning discriminative features from face images to facilitate accurate and efficient recognition.

The document's emphasis on face space and the challenge of recognizing faces under varying conditions directly relates to the motivation behind the Siamese network project. Siamese networks excel at learning representations that capture facial similarities and differences, making them suitable for addressing the limitations of traditional methods like eigenfaces. Specifically, Siamese networks can handle variations in lighting, scale, and orientation more effectively through their ability to learn feature embeddings that are robust to such changes.

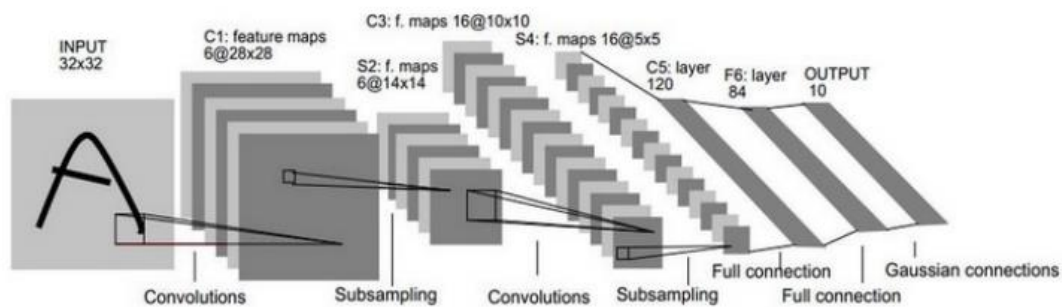
This work demonstrates the gaps that the Siamese network project can fill by showcasing the limitations of the eigenface approach. Eigenfaces, while effective in certain scenarios, struggle with recognition in non-ideal conditions. Siamese networks offer a solution by learning representations that can adapt to these variations, resulting in improved recognition performance and more accurate face matching.

From this literature, several ideas can be derived for the Siamese project. The concept of face space and the idea of projecting faces into a subspace of known individuals can guide the development of Siamese networks to create compact and discriminative embeddings. Additionally, the notion of adaptability in the face of changing conditions can inspire the inclusion of augmentation techniques and robust training strategies within the Siamese network project.

The credibility of this source lies in its academic nature, as it is likely sourced from a research publication given its detailed experiments, references to other works, and the inclusion of recognized researchers in the field. The paper cites well-known methodologies such as PCA and eigenfaces, and the experimental approach and results add to its reliability.

Face recognition using Convolution Neural Network (CNN)

The document titled "Research on Face Recognition Based on CNN" explores the utilization of Convolutional Neural Networks (CNNs) in the field of face recognition. The study investigates the fundamental principles of CNNs, focusing on constructing convolutional and downsampled layers using OpenCV functions to process images. The article also delves into the concepts of Multi-Layer Perceptron (MLP) to comprehend fully connected and classification layers, using the Theano library in Python for implementation. The authors simplify the CNN model by merging convolutional and sampling layers, which significantly enhances image recognition rates. (Jie Wang and Zihao Li ,2018)



(Figure 2, Convolution Neural Networks Architecture)

The study establishes the significance of CNN in face recognition, specifically for embedding models in a Siamese network. A Siamese network involves training on pairs of images to determine whether they are of the same identity or not. The CNN discussed in the paper serves as a strong candidate for generating embeddings, as its hierarchical feature extraction aligns well with the concept of creating distinguishable features for image pairs. CNNs can learn intricate patterns and features, which are crucial for accurate face recognition.

The document highlights the limitations of traditional identification methods and underscores the potential of CNN-based face recognition. It acknowledges that deep learning algorithms, like CNN, require a substantial number of samples to construct a robust model. This limitation creates a gap that a Siamese network can fill, as it operates efficiently with limited data by comparing pairs of images, rather than relying solely on a large dataset. The CNN's ability to extract features effectively serves as motivation for integrating it into a Siamese project, which aims to address the challenge of face recognition with scarce data.

The research's application of CNN in face recognition aligns with the broader societal need for accurate, automated, and secure identification systems. The study's findings can pave the way for advancements in security systems, access control, surveillance, and more. The integration of CNN within a Siamese network offers innovative potential, particularly in scenarios where data availability is limited. This approach can benefit law enforcement, public safety, and various sectors that require robust yet efficient face recognition systems.

The CNN's ability to extract distinctive features can greatly enhance a Siamese network's performance. By employing the learned features as embeddings, the Siamese network can effectively measure the similarity between pairs of faces, regardless of variations in lighting, pose, and expression. The hierarchical nature of CNNs aligns with the layered structure of Siamese networks, making the integration intuitive and potentially leading to improved accuracy in pairwise comparisons.

This document is in the IOP Conference Series, known for strict peer review and scholarly work. The authors, Jie Wang and Zihao Li, are from the School of Electrical Engineering at Zhengzhou University, showing their knowledge in the field. They use references to build on existing research, and they provide a detailed methodology, results, and references to previous studies, making their source trustworthy.

Face Recognition using The Viola-Jones Algorithm

The document provided contains a comprehensive literature review on the Viola-Jones algorithm and its applications in various fields, including face detection, emotion recognition, and driver assistance systems. (M. F. Hirzi, S. Efendi and R. W. Sembriring, 2021) The credibility of this source is demonstrated through the extensive list of references from reputable journals, conferences, and research papers in the field of computer vision and machine learning, ensuring that it draws from well-established research.

The literature reviewed in this document contributes significantly to a Siamese network project by highlighting the strengths and weaknesses of the Viola-Jones algorithm, which serves as a benchmark technique in face detection and recognition. It offers insights into the algorithm's accuracy, precision, and efficiency in different contexts, such as real-time face recognition and emotion detection. This information is valuable for a Siamese network project, as it helps identify areas where Siamese networks can potentially enhance the performance of face-related tasks.

Moreover, the literature review demonstrates gaps that a Siamese network project can address. For instance, the document points out limitations in the Viola-Jones algorithm's performance, such as its inability to handle variations in facial expressions or poses. These gaps highlight opportunities for Siamese networks to excel, as they are known for their robustness in handling such variations.

The document references several research studies that assess the effectiveness of the Viola-Jones algorithm in various applications, including face detection, emotion recognition, and driver drowsiness detection. These studies provide evidence of the algorithm's capabilities and its limitations, which can serve as a valuable foundation for a Siamese network project seeking to improve upon these areas.

Furthermore, the literature reviewed here motivates the Siamese network project by showcasing the potential benefits for society. For instance, the application of Siamese networks in driver assistance systems, as demonstrated in some of the referenced studies, can significantly enhance road safety by accurately detecting driver drowsiness or inattention. This motivation aligns with the broader societal goal of reducing accidents and improving transportation safety.

CHAPTER 3:PROJECT DESIGN

Domain and Users

The scope of this project revolves around the creation of an innovative attendance system tailored for school environments. The main focus is on developing an advanced system that optimizes attendance tracking. The primary beneficiaries of this system are students, who stand to gain significantly from a more streamlined and precise method of recording their attendance. This is particularly relevant due to students' familiarity with such technological solutions and their readiness to embrace them.

An additional rationale for this approach is the necessity for the school to maintain a repository of student images, which can serve as training data for the underlying model's recognition capabilities. This aids in enabling the system to accurately identify students. Nonetheless, it's important to note that the system's user base isn't restricted solely to students; it extends to employees within the organization as well as regular customers frequenting unmanned stores. This versatility underscores the system's potential impact across various contexts.

Design Choices

This project comprises two key phases: the embedding phase and the classification phase. The primary objective of the embedding phase, implemented through a Siamese network, is to transform images into structured vectors that can be readily comprehended by a machine. These vectors are subsequently input into the classification phase, which employs the K-Nearest Neighbors (KNN) algorithm for the purpose of facial recognition.

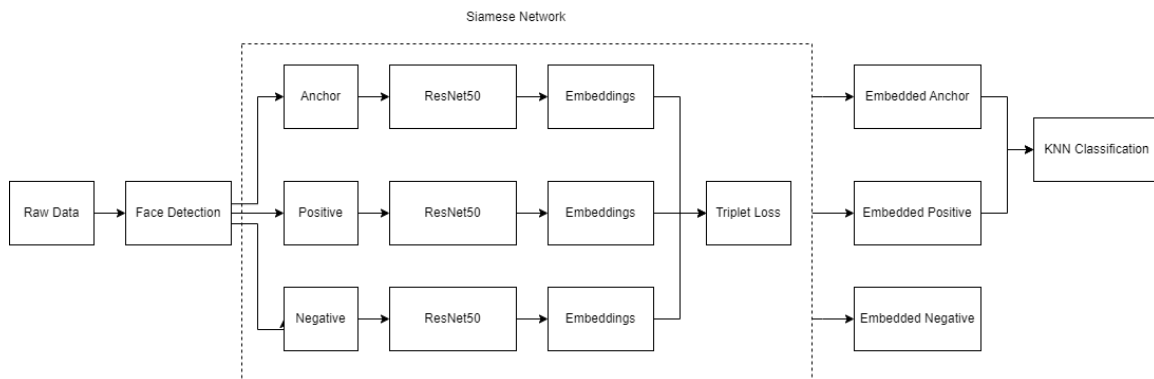
The choice of a Siamese network for the embedding model is underpinned by its robustness. This type of network is specifically designed to learn how to bring similar images closer together while pushing dissimilar images apart. To achieve this, a triplet model architecture is employed, which represents an enhanced version of the Siamese network. Within this architecture, the model is trained to recognize that data from a positive input (i.e., images of the same individual) is similar to the anchor, while data from a negative input (i.e., images of different individuals) is dissimilar to the anchor. This approach is particularly beneficial when dealing with a limited number of images per student, as it enables the model to learn and distinguish between individual students through similarity comparisons. In contrast, a deep neural network is unsuitable for this scenario since it necessitates a large number of images for each student to capture their unique features effectively.

In the classification phase, KNN is employed for the specific purpose of classification. The model is trained to identify the nearest neighbors in the embedding space. It is trained on embedded student faces, and when faced with unseen embedded data, it can effectively locate the nearest neighbors for classification.

Overall Structure

This project involves several steps, beginning with the collection of raw data from a dataset. This data undergoes preprocessing, including face detection to remove background noise and resizing to match the input requirements of the embedding model. The preprocessed data is then stored in a new directory, and a triplet is generated for each data point. These triplets are subsequently fed into the Siamese network, which processes them through the ResNet50 and embeddings model.

During the training process, a triplet loss is computed and continually optimized to minimize the loss value. The ultimate output of the Siamese network is an embedded triplet. The embedded anchor and embedded positive are combined and utilized for K-nearest neighbors (KNN) training. This is because they have been trained to be closer to each other during the Siamese training process, making them suitable for KNN-based tasks. The architecture of the embedding model can be found in [Appendix A](#).



(Figure 3, System Architecture)

Technologies and Methods

Machine Learning

A Siamese network is a powerful unsupervised learning technique that excels even when dealing with unlabeled datasets. During the training process, the network is provided with a dataset that exclusively consists of triplets. While this dataset may lack labels, the system still discerns that the anchor input and positive input must represent the same entity, while the negative input signifies a different one.

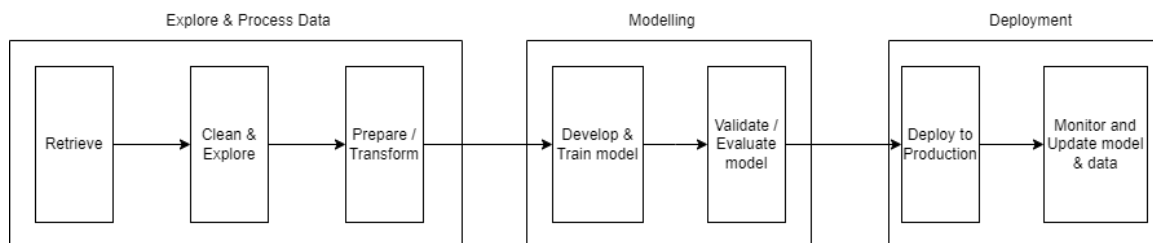
To achieve this, a custom loss function known as triplet loss is employed. The model employs the backpropagation algorithm, allowing for forward propagation and iteratively adjusting its weights to minimize the loss value. This process is highly significant because as the loss value decreases, the distance between the anchor and positive inputs shrinks, while the distance between the anchor and negative inputs increases. Consequently, this enables the network to group images more accurately, ultimately enhancing its performance.

Computer Vision

This project consists of two computer vision tasks: face detection and converting pictures into numbers. When collecting raw data, many of the pictures contain backgrounds or multiple people. Therefore, we need to utilize the OpenCV library to detect faces in the pictures and extract them to a separate location. If multiple faces are detected, the picture should be ignored. Additionally, OpenCV is employed to read the input pictures and convert them into numerical data by extracting their pixel values. This step is crucial as it enables computers to accurately interpret the pictures and perform further preprocessing.

Universal Workflow of Machine Learning

Machine learning encompasses a variety of essential concepts, with the universal workflow serving as a foundational framework applicable to all machine learning endeavors. This comprehensive approach comprises seven crucial steps, thoughtfully ordered to guide the resolution of any machine learning challenge. Initially, it involves defining the problem at hand and assembling a relevant dataset, followed by the critical task of selecting a measure of success. Subsequently, the workflow involves determining an appropriate evaluation protocol, preparing the data meticulously, and embarking on the development of a model that surpasses a baseline performance. As the project progresses, the focus shifts to scaling up the model, intentionally pushing it to the point of overfitting. Lastly, the workflow emphasizes the vital processes of regularization and hyperparameter tuning. These sequential steps serve as a steadfast guide throughout the project's evaluation and analysis, ensuring a comprehensive and methodical approach to machine learning tasks.



(Figure 4, Machine Learning Workflow)

Data Augmentation

This process encompasses a spectrum of transformative operations such as rotation, flipping, and zooming, all of which yield diverse iterations of the original images. By introducing subtle modifications, we can generate multiple analogous images, thereby amplifying the image count significantly.

Python

This project will be implemented using Jupyter Notebook, a highly user-friendly platform that provides valuable resources for machine learning purposes. Python has been chosen as the programming language due to its simplicity and wide range of excellent libraries and frameworks available for AI and machine learning.

Work Plan

The work plan will be shown through a Gantt Chart at [Appendix B](#).

Testing and Evaluation Techniques**Embedding model**

Multiple embedding models will be created with varying architectures, including deep and custom neural network structures and pretrained models functioning as feature extractors with custom embedding layers. These models will be tested using different batch sizes, such as 64 and 32.

Evaluation of these models will focus on loss and validation loss metrics rather than accuracy, as the primary purpose of these models is not classification or prediction. A lower loss value indicates better model performance. Loss value is computed based on the training dataset, while validation loss is calculated using a separate validation dataset, often referred to as an unseen dataset.

Classification model

In the evaluation of this model, several techniques will be employed to assess its performance comprehensively. The Confusion Matrix will be used to gain insights into how the model performs for each individual class or category. The F1 Score, which is the harmonic mean of precision and recall, will provide an overall understanding of the model's prediction errors. We will also consider the Accuracy of the model, which measures its ability to correctly predict the true identity or category of data points. Finally, the AUC-ROC (Area Under the Receiver Operating Characteristic Curve) will be employed as a performance metric, especially useful for classification problems, to assess the model's capability to distinguish between different classes across various thresholds.

CHAPTER 4: IMPLEMENTATION

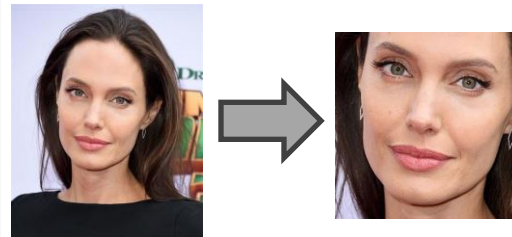
Defining the problem and processing data

In this project, three different face detection models are tested: Haar Cascade, MTCNN, and Dlib. The Haar Cascade model utilizes binary classification and edge detection techniques to analyze pixels in an image for the presence of facial features like skin, eyes, and mouth. In contrast, the MTCNN model employs a three-stage neural network to identify and classify faces with a higher accuracy. Lastly, the Dlib method extracts gradient distributions from images, converts them into a Histogram of Oriented Gradients (HOG), and uses a support vector machine (SVM) as a classifier.

```
# detect face from the image
faces = detector(gray_image)

if len(faces) == 0:
    print('No faces detected at ', root.split('\\')[-1], ", ", file)
    empty += 1
    continue

# iterate through the face detected
for face in faces:
    # only 1 face should be detected
    if len(faces) == 1:
        # read the position of face
        x, y, x1, y1 = face.left(), face.top(), face.right(), face.bottom()
```

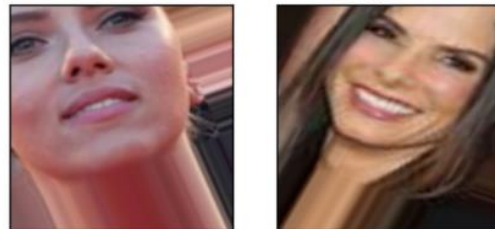


(Figure 5, Face Detection code)

(Figure 6, Face Detection result)

To maximize the utilization of the images in the dataset, the data augmentation technique is applied using the ImageDataGenerator from the TensorFlow Keras preprocessing library. This technique involves generating five augmented versions of each image, resulting in a total dataset size of 9,000 images. This approach enhances the dataset's diversity and mitigates the risk of model overfitting.

```
for batch in datagen.flow(x, batch_size=1):
    augmented_img = batch[0].astype(np.uint8)
    images.append(augmented_img)
    labels.append(celebs)
    i += 1
    if i >= 5: # Generate 5 augmented images
        break
```



(Figure 7, Data Augmentation code)

(Figure 8, Data Augmentation result)

A triplet typically consists of an anchor, a positive, and a negative element. The process of generating a triplet involves iterating through each image and storing it as an anchor in an array. The class of the anchor image is recorded, and then another image from the same class is randomly selected as the positive. For the negative element, an image is randomly chosen from a class different from the recorded class.

```

A,P,N = [],[],[]
numClasses = len(np.unique(labels))
# index of labels for each class
idx = [np.where(labels == i)[0] for i in np.arange(0, numClasses)]
for i in np.arange(len(images)):
    # store current images
    A.append(images[i])
    label = labels[i]
    # positive images (same class with anchor)
    P_index = np.random.choice(idx[label])
    P.append(images[P_index])
    # negative images (different class with anchor)
    N_index = np.random.choice(np.where(labels != label)[0])
    N.append(images[N_index])

```

(Figure 9, Triplet generator code)



(Figure 10, Triplet example)

Feature Engineering

In this project, three different feature engineering methods were tested. The first approach involved the creation of a custom deep neural network, developed without the use of pretrained models. The second method made use of the pretrained model served solely as a feature extractor to capture meaningful data features. Additionally, the output layers of the embeddings were customized to further optimize model performance. To ensure compatibility with the selected model architecture, the input data was resized to dimensions of (224, 224, 3).

Siamese model A: Custom deep neural networks

This model is the initial version of Siamese model where the embedding model is customized. It starts with two sets of Conv2D layers with 64 filters each, followed by max-pooling layers to downsample the feature maps. This pattern is repeated with increasing filter sizes (128, 256, 512) and additional max-pooling layers for hierarchical feature extraction. Dropout layers are included for regularization. Then, there are two more Conv2D layers with 1024 filters each, followed by max-pooling and dropout layers. The last convolutional layer has 4096 filters with a dropout layer. The model then globally averages the feature maps, adds a dense layer with 1024 units, and finally, outputs a dense layer with 128 units for classification. The model uses ReLU activation functions for all convolutional layers and does padding to maintain the spatial dimensions of the feature maps.

Siamese model B: Resnet50 as feature extractor + embeddings output layers

The model starts with an input layer that takes input data with a specified shape. The input data is pre-processed using the ResNet50 pre-processing function. Then, a pre-trained ResNet50 model with weights from ImageNet is loaded and frozen, meaning its weights won't be updated during training. The input data is passed through this base CNN, extracting features. After that, the extracted features are processed through several trainable layers: a global average pooling layer, followed by three dense layers with decreasing units, ReLU activation, dropout for regularization, and batch normalization

for stability. The final output layer produces embeddings of size 128. This model is designed for feature extraction and embedding generation.\

Siamese model C: VGG16 as feature extractor + embeddings output layers

Compared to ResNet-50, VGG16 boasts a significantly larger number of parameters. While this increased complexity can make VGG16 capable of extracting a richer set of features and acquiring more meaningful embeddings than ResNet-50, it's important to note that this advantage comes at the cost of computational resources and potentially increased training time.

Siamese model D: InceptionV3 as feature extractor + embeddings output layers

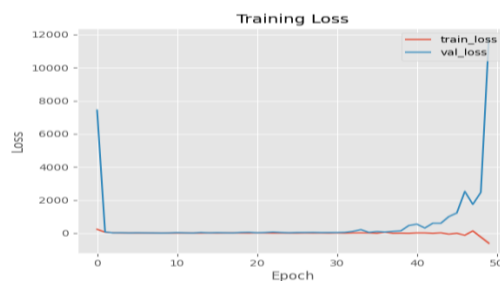
InceptionV3 is a model closely related to ResNet50, yet it exhibits greater efficiency in general. It excels in capturing multi-scale features, making it particularly well-suited for scenarios like this one, where numerous distinct classes are encountered.

Overfitting

Upon evaluation, the model exhibited clear signs of overfitting, necessitating a series of refinements. To address this issue, the dropout rate was increased from 0.3 to 0.5, introducing more regularization to the model. Additionally, early stopping was implemented to prevent overfitting behaviors by halting training when validation loss did not show improvement for 5 epochs. Furthermore, experiments were conducted with different batch sizes, including 32 and 64, to discern which one would yield superior results. These adjustments collectively aimed to bolster the model's robustness and generalization capabilities, ultimately enhancing its overall performance.

```
# Create an EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitor
    patience=5,         # Number
    restore_best_weights=True # Res
```

(Figure 11, Early Stopping code)



(Figure 12, Overfitting result)

Classification model - K-nearest neighbors (KNN)

The trained embedding model is employed to embed triplets by predicting functions. The embedded anchor is subsequently concatenated with the embedded positive. These embedded training pairs are then fed into the KNN model for training purposes, while the embedded testing pairs are utilized for evaluation. The evaluation process involves comparing the similarity between predictions and true labels.

CHAPTER 5: EVALUATION

Face Detection model

The total number of faces detected by each algorithm has been meticulously recorded and is presented below. Among these results, the Dlib algorithm shines as it demonstrates the capability to identify the highest number of faces, a remarkable 1715 in total, making it the preferred choice. Following closely is MTCNN, which impressively detects 1689 faces, and Haar Cascade, which still performs well with 1460 faces detected.

Number of images	Dlib	MTCNN	Haar Cascade
1800	1715	1689	1460

(Table 1, Face detection model result)

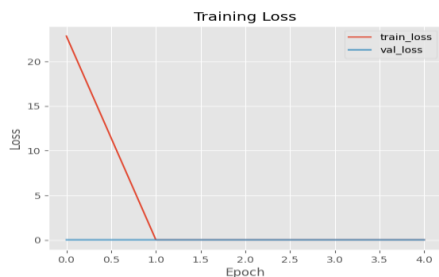
Baseline (Naïve model)

Calculating the likelihood of accurately classifying a celebrity's identity without relying on extensive computational methods, commonly referred to as a random guess, involves making a name selection at random, devoid of any analytical processes. In the dataset, there are 17 distinct celebrities. Therefore, when selecting a celebrity name at random, the probability of guessing correctly is $1/17$, equivalent to 0.0588 or approximately 5.88%. This 5.88% probability serves as baseline for improving classification accuracy.

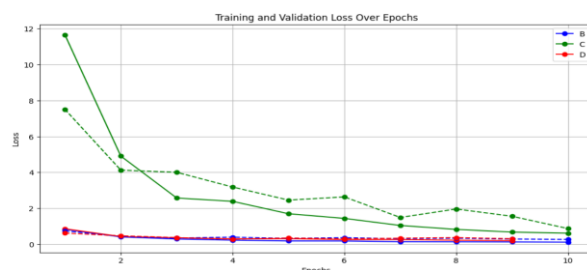
Siamese embedding model

The models are evaluated by tracking their loss and validation loss values during training. Model A's extremely low loss value near zero may seem impressive, but it's misleading because the model isn't designed well and doesn't show genuine learning.

Models B, C, and D reveal more interesting insights. Model C starts with the highest loss value but consistently decreases, indicating learning progress. Models B and D initially perform similarly, but Model D's progress stalls around epoch 9, suggesting overfitting—learning the training data too well but struggling to adapt to new data. Therefore, ResNet50, Model B, emerges as the most suitable feature extractor due to its stable performance and avoidance of overfitting. Individual model loss graph at [Appendix C](#).

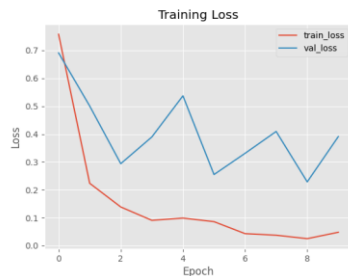


(Figure 13, model A evaluation)

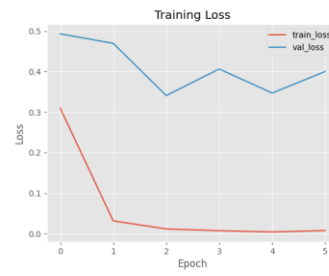


(Figure 14, model B, C, D evaluation)

The embedding model was tested with batch sizes of 32 and 64. The 64-batch size model performed poorly on validation loss, while the 32-batch size model showed a decreasing trend in validation loss. This suggests that the smaller batch size is exposed to more diverse examples thus do not overfit. Therefore, we opted for a batch size of 32.

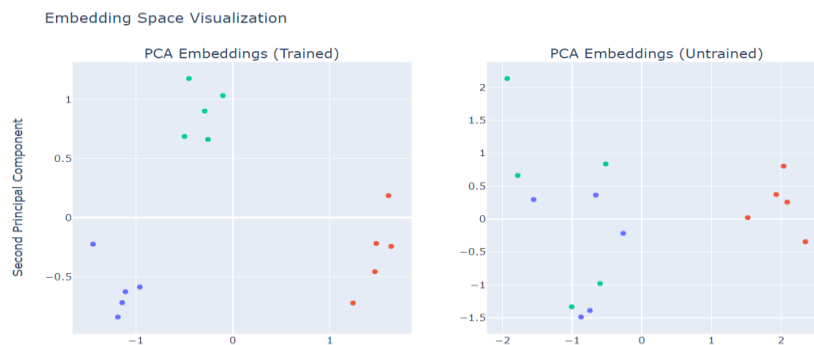


(Figure 15, 32 batch size)



(Figure 16, 64 batch size)

The figure below illustrates the results of embedding obtained from both a trained and an untrained model. The untrained model, shown on the right, struggles to effectively segregate data points into distinct categories. Conversely, the trained model on the left demonstrates clear separation among three distinct categories represented by the colors red, green, and blue, and these categories are noticeably distant from one another. This provides a response to the first question, "How does a Siamese network contribute to face recognition?". Additional feature visualizations can be accessed in [Appendix D](#).



(Figure 17, Embedding Space Visualization)

KNN classification model

Based on Figure 14, it is evident that there are only two machine learning methods deemed suitable for classification purposes: KNN and Support Vector Machines (SVM). However, it is worth noting that this dataset encompasses a total of 17 classes, making it considerably complex. SVM, traditionally, may encounter challenges in effectively handling multi-class classification tasks. Therefore, KNN stands out as the most suitable option because it is highly likely that the nearest neighbor belongs to the same class. Additionally, this choice also addresses the second question, "Which classification method should be employed to accurately identify each individual?"

Evaluation metrics	Accuracy	Recall	Precision	F1-score
KNN scores	53%	53.1%	52.9%	53.7%

(Table 2, KNN Classification model result)

The model has been evaluated using several key performance metrics. It achieves an accuracy of 53%, indicating that it correctly predicts the class labels for slightly more than half of the instances. The recall, which measures its ability to identify positive instances, stands at 53.1%, while the precision, indicating the accuracy of its positive predictions, is 52.9%. The F1-score, a balanced measure of performance, is 53.7%, suggesting a reasonably balanced trade-off between precision and recall. Finally, it is evident that the classification model has surpassed the baseline model in terms of accuracy. Further detailed evaluation results can be found in [Appendix E](#).

Advantages

This approach boasts exceptional computational speed, with the entire process revolving around three key steps: detection, embedding, and classification. In contrast to the methodology described in Literature Review 3, where images are fed into deep neural networks, a significantly more time-consuming endeavor, this approach stands out. Moreover, it stands out in its efficiency as it does not necessitate an extensive volume of training data. This is because it does not learn individualized features for each person but instead focuses on mastering the effective embedding of each individual.

This characteristic makes the model highly suitable for attendance tracking purposes, particularly in situations where a multitude of students check in and out on a daily basis. In such scenarios, a rapid process is essential, and this approach delivers on that requirement. Additionally, the school typically maintains a relatively limited set of pictures exclusively for the students which is enough to train the model.

Disadvantages

This approach has several significant drawbacks. Firstly, complex datasets with numerous features pose a primary challenge, as KNN's ability to reliably identify the correct answer diminishes in the face of such intricacy. Secondly, KNN heavily relies on the similarity of data points for predictions. This becomes problematic when dealing with highly similar data points, as seen in cases like facial recognition with identical twins, where KNN may struggle to distinguish between them. The efficacy of KNN hinges on the quality of the embedding or feature extraction techniques used to differentiate such closely resembling instances. Lastly, another limitation of KNN lies in the significant variability of accuracy across different individuals within the dataset. This inconsistency makes it arduous to ensure consistently high accuracy for each individual or category.

CHAPTER 6: CONCLUSION

This project focuses on achieving precise human face identification for an attendance marking system using a dataset comprising 17 celebrity faces. It involves experimentation with various models and parameters aimed at enhancing the system's architecture, ultimately resulting in the selection of the most effective model. The implementation of this project within an educational institution would significantly enhance the accuracy of student attendance tracking. This is due to the requirement for students to be physically present in the classroom in order to mark their attendance, eliminating the possibility of cheating.

The project initially started with a baseline model achieving an accuracy of 5.88%. To enhance the face recognition pipeline, a combination of deep learning techniques was employed for the Siamese model, while machine learning techniques were utilized for the K-Nearest Neighbors (KNN) approach. Several measures were implemented to address and mitigate the issue of overfitting. Through these efforts, the model was substantially improved, ultimately achieving an impressive accuracy rate of 53%. This remarkable accomplishment represents nearly a tenfold increase in accuracy when compared to the baseline model.

- How does a Siamese network contribute to face recognition?
- Which classification method will be employed to accurately identify each individual?

Within the project's scope, two fundamental questions have been successfully addressed. Firstly, a Siamese network is employed as the embedding model for processing human faces. This network processes input faces, subjecting them to a series of embedding processes that yield embedded representations of these faces. Secondly, the KNN algorithm emerges as the preferred classification method. This choice is because the embedded faces in the embedding space display a noticeable degree of separation. This characteristic makes the algorithm particularly well-suited, as it excels at measuring distances within individual clusters, enhancing its suitability for the task at hand.

Further work

In this project, we have successfully developed a comprehensive face recognition pipeline. However, it's important to note that this pipeline is not fully automated. To enhance its practicality and real-world applicability, we can transform this model into an AI-as-a-Service Operating Model that can seamlessly integrate into fully developed applications as an additional feature, thereby contributing significantly to real-world use cases.

Furthermore, the evaluation results have highlighted areas where improvements are needed. One key approach to enhance the model's performance is by training it using a deeper embedding model, leveraging the increased computing resources available. This strategic adjustment aims to reduce the loss value and facilitate the convergence of anchor points towards their respective positive samples. By achieving a higher degree of proximity between anchors and their designated positives, we can substantially improve the accuracy of the classification model when identifying nearest neighbors. This improvement will have a direct positive impact on the model's overall performance.

Broader themes

The core strength of this model lies in its ability to discern patterns and similarities within data, irrespective of the data type or context. For instance, it can excel in the complex task of recognizing traffic signs, a crucial component of autonomous driving systems. By learning the shared characteristics and features among various traffic signs, this model can contribute to enhancing road safety and navigation.

Moreover, the versatility of this concept extends to household recognition for robotics applications. Whether it's identifying everyday objects or understanding the layout of a domestic environment, the model's capacity to extract similarities between data points can significantly improve the capabilities of household robots. This may involve tasks such as locating items, navigating through cluttered spaces, or even assisting with household chores.

In essence, the potential applications of this model reach far beyond face recognition, offering a powerful tool for understanding and processing diverse datasets in a wide array of fields. Its strength lies in its ability to decipher the hidden connections within data, making it an invaluable asset for innovation and problem-solving across industries.

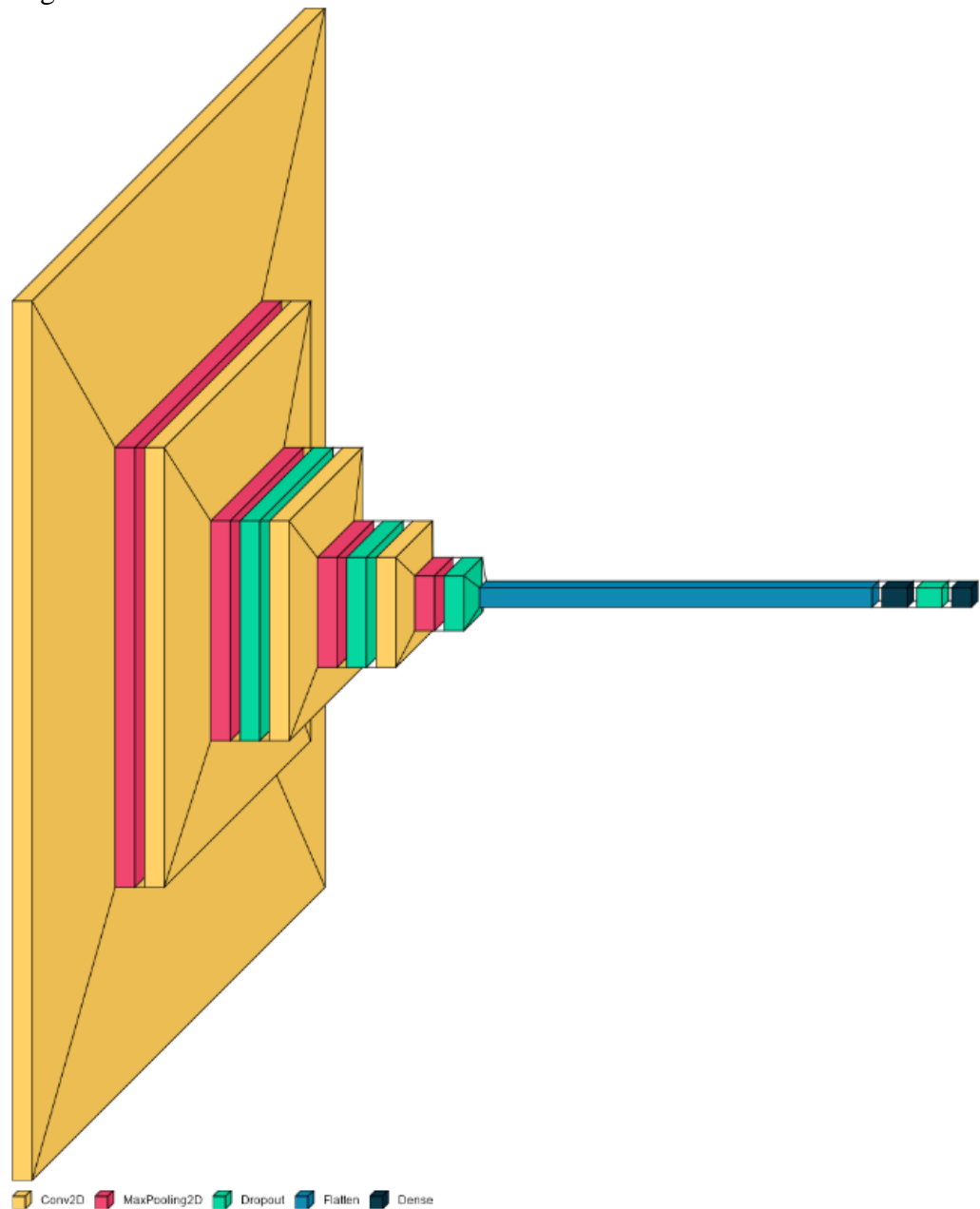
CHAPTER 7: APPENDICES

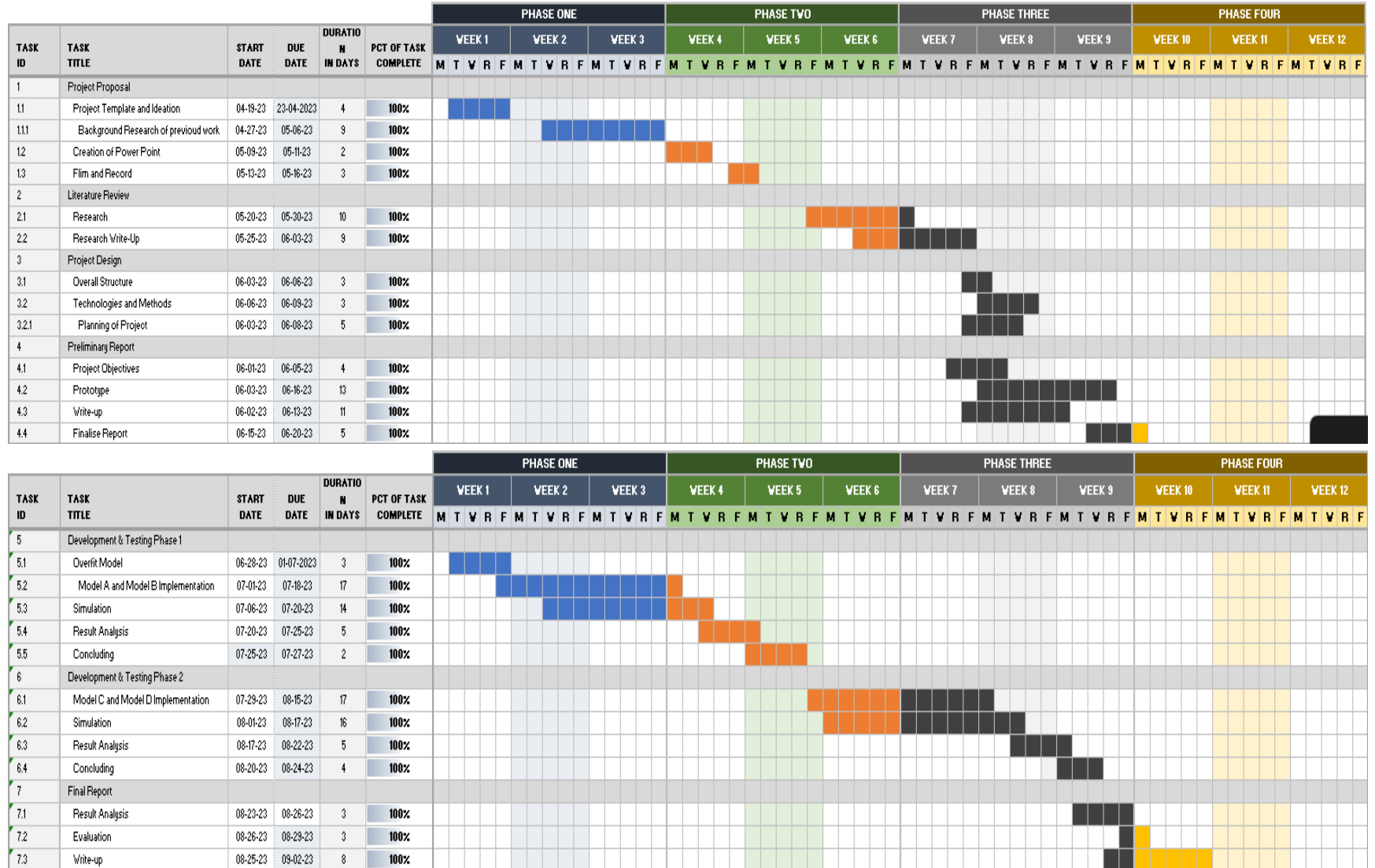
Link to code repository:

https://drive.google.com/file/d/18WUugw0alZ6wJxnRggJn16v1Dpk6ZS_j/view?usp=drive_link

A. Embedding model

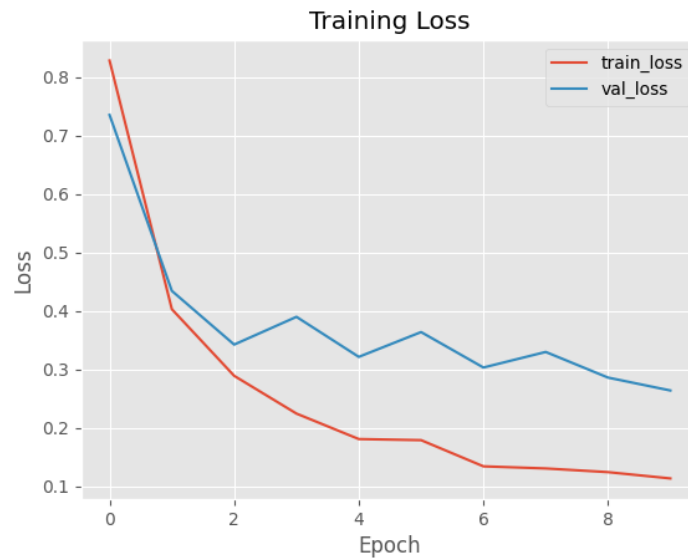
Out[12]:



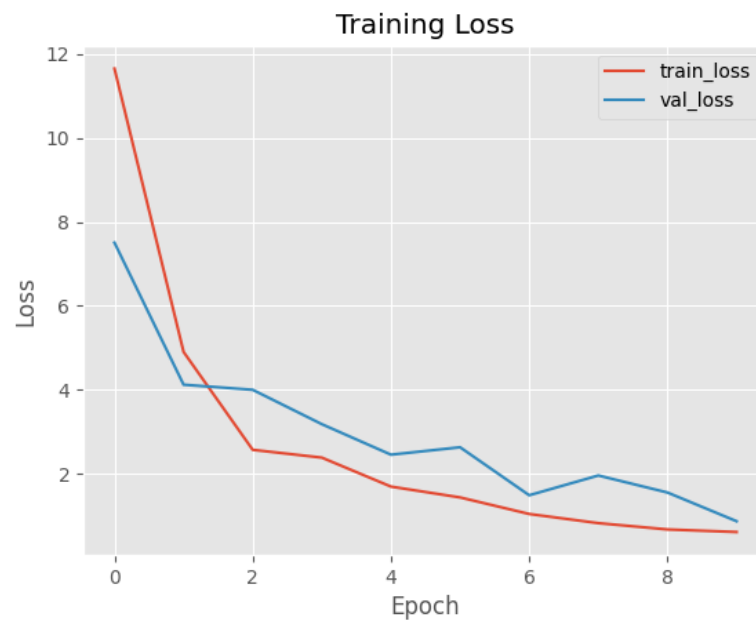


C. Graph Depicting Loss for Each Siamese Model

1. Model B loss graph



2. Model C loss graph

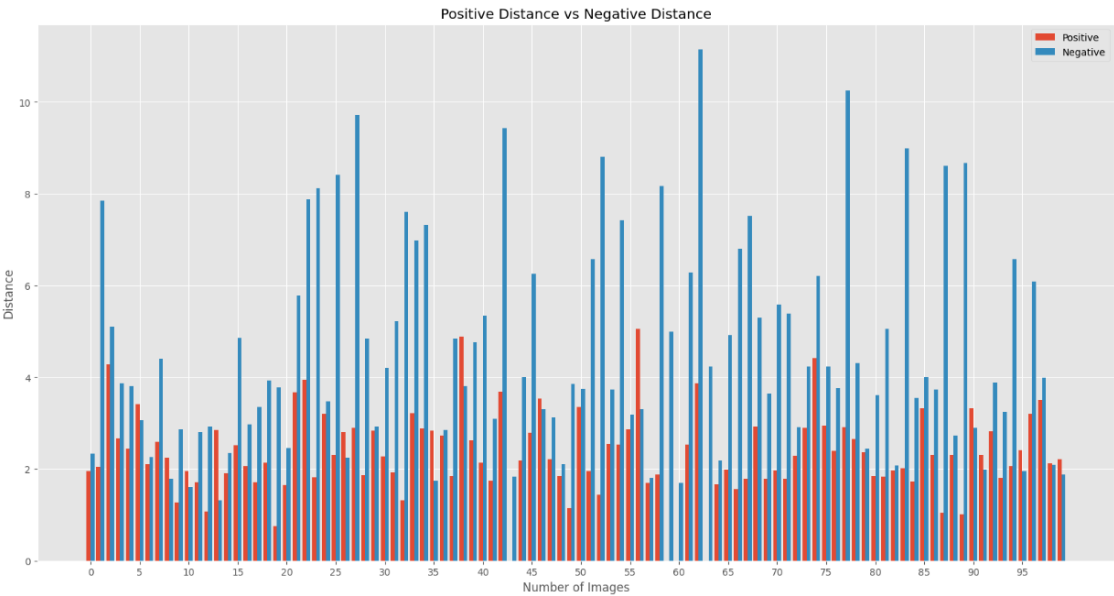


3. Model D loss graph

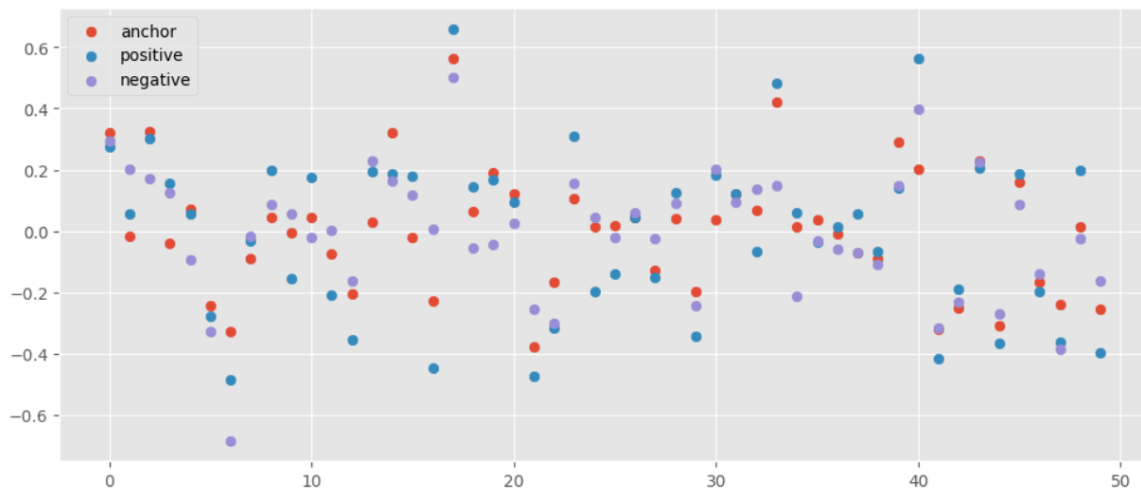


D. Feature visualization

1. Positive distance against negative distance



2. Distance between anchor, positive and negative

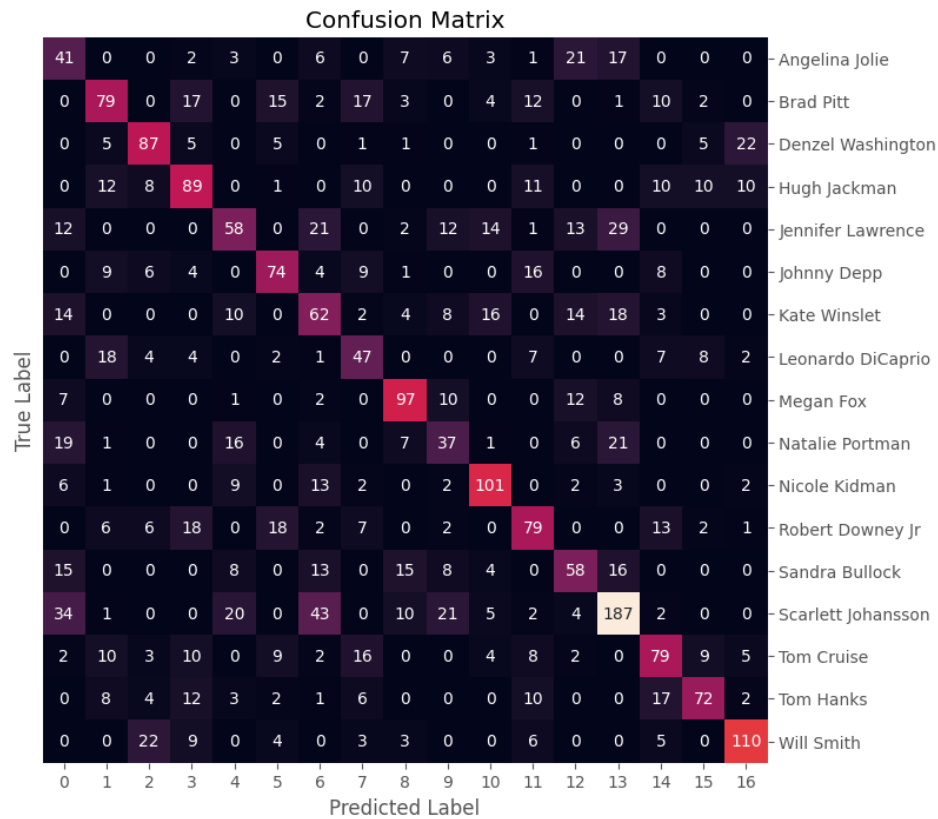


E. KNN Evaluation metrics

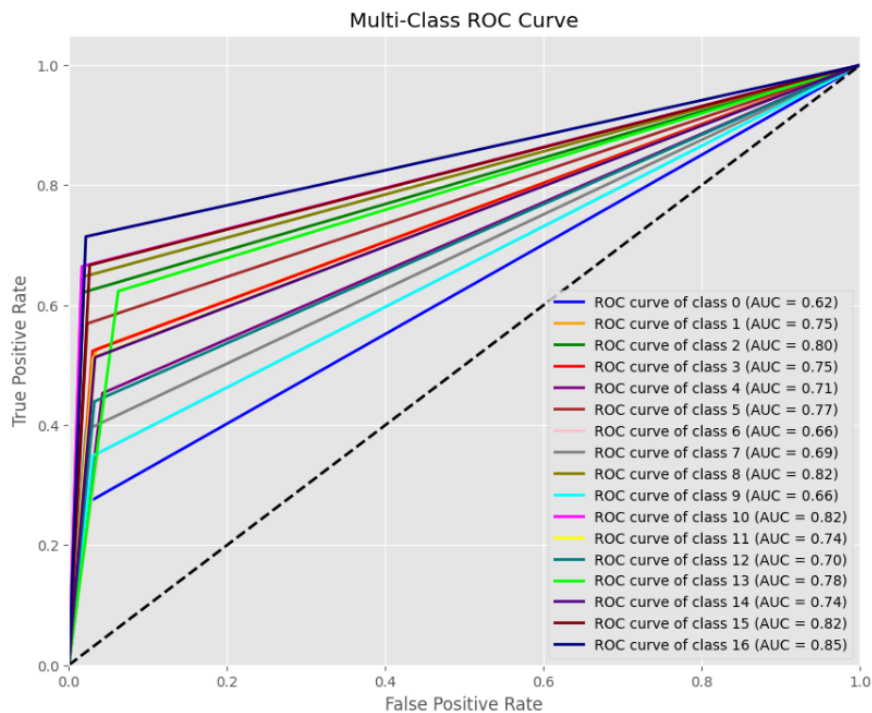
1. Classification report

	precision	recall	f1-score	support
0	0.38	0.27	0.32	150
1	0.49	0.53	0.51	150
2	0.66	0.62	0.64	140
3	0.55	0.52	0.54	170
4	0.36	0.45	0.40	128
5	0.56	0.57	0.57	130
6	0.41	0.35	0.38	176
7	0.47	0.39	0.43	120
8	0.71	0.65	0.68	150
9	0.33	0.35	0.34	106
10	0.72	0.66	0.69	152
11	0.51	0.51	0.51	154
12	0.42	0.44	0.43	132
13	0.57	0.62	0.59	300
14	0.50	0.51	0.50	154
15	0.53	0.67	0.59	108
16	0.68	0.71	0.70	154
accuracy			0.53	2574
macro avg	0.52	0.52	0.52	2574
weighted avg	0.53	0.53	0.53	2574

2. Confusion matrix



3. ROC Curve




CHAPTER 8: REFERENCES

Citation Type: Harvard Referencing

1. NEC (2020). A brief history of facial recognition - NEC New Zealand. Available at: <https://www.nec.co.nz/market-leadership/publications-media/a-brief-history-of-facial-recognition/#:~:text=The%20dawn%20of%20Facial%20Recognition,to%20recognise%20the%20human%20face>.
2. Halime Yilmaz (2022) Top 5 use cases of face recognition, Cameralyze. Available at: <https://www.cameralyze.co/blog/top-5-uses-of-face-recognition-in-2022>.
3. P. Jonathon Phillips. Support Vector Machines Applied to Face Recognition. Available at: <https://proceedings.neurips.cc/paper/1998/file/a2cc63e065705fe938a4dda49092966f-Paper.pdf>
4. Matthew A. Turk and Alex P. Pentland. Face Recognition Using Eigenfaces. Journal of Cognitive Neuroscience, March 1991. Available at: <https://sites.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf>
5. Jie Wang and Zihao Li (2018). Research on Face Recognition Based on CNN. IOP Conference Series: Earth and Environmental Science, 170(3), 032110. doi:10.1088/1755-1315/170/3/032110. Available at: <https://iopscience.iop.org/article/10.1088/1755-1315/170/3/032110/pdf>
6. Noufal (2020) Triplet loss: CNN Model + KNN first attempt, Kaggle. Available at: <https://www.kaggle.com/code/kvsnoufal/triplet-loss-cnn-model-knn-first-attempt/notebook>
7. Chandhok, S. (2023) Face recognition with Siamese networks, Keras, and tensorflow, PyImageSearch. Available at: <https://pyimagesearch.com/2023/01/09/face-recognition-with-siamese-networks-keras-and-tensorflow/>.
8. Agarwal, V. (2020) Face detection models: Which to use and why?, Medium. Available at: <https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c>.
9. Stoicstatic (2021) Face recognition: Siamese w/ Triplet Loss, Kaggle. Available at: <https://www.kaggle.com/code/stoicstatic/face-recognition-siamese-w-triplet-loss/notebook>.
10. Chandhok, S. (2023b) Training and making predictions with Siamese networks and triplet loss, PyImageSearch. Available at: <https://pyimagesearch.com/2023/03/20/training-and-making-predictions-with-siamese-networks-and-triplet-loss/>.
11. Building a facial recognition pipeline with deep learning in tensorflow HackerNoon. Available at: <https://hackernoon.com/building-a-facial-recognition-pipeline-with-deep-learning-in-tensorflow-66e7645015b8>.

12. Tf.keras.model : tensorflow V2.13.0 TensorFlow. Available at:
https://www.tensorflow.org/api_docs/python/tf/keras/Model#used-in-the-notebooks_1.
13. Multiclass receiver operating characteristic (ROC) (no date) scikit. Available at:
https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py.
14. Chandhok, S. (2023) Triplet loss with Keras and tensorflow, PyImageSearch. Available at: <https://pyimagesearch.com/2023/03/06/triplet-loss-with-keras-and-tensorflow/>.
15. Team, K. (no date) Keras Documentation: Image similarity estimation using a Siamese network with a triplet loss, Keras. Available at:
https://keras.io/examples/vision/siamese_network/.
16. Gupta, M. (2023) Understanding siamese network with example and codes, Medium. Available at: <https://medium.com/data-science-in-your-pocket/understanding-siamese-network-with-example-and-codes-e7518fe02612>.
17. Strange Travel (2023) Facial_recog_siamese_network, Kaggle. Available at:
<https://www.kaggle.com/code/strangetravel/facial-recog-siamese-network>.
18. Abdou, A. (2022) Introductory guide for triplet loss function, Ahmed Abdou's Blog. Available at: <https://ahmedabdou.hashnode.dev/introductory-guide-for-triplet-loss-function>.
19. Khandelwal, R. (2021) Siamese network for facial recognition, Medium. Available at: <https://medium.com/swlh/siamese-network-for-facial-recognition-5bd33be9e381>
20. J. Rafid Siddiqui, P. (2021) Latent space representation: A hands-on tutorial on autoencoders using TENSORFLOW, Medium. Available at:
<https://medium.com/mlearning-ai/latent-space-representation-a-hands-on-tutorial-on-autoencoders-in-tensorflow-57735a1c0f3f>.
21. M. F. Hirzi, S. Efendi and R. W. Sembiring, "Literature Study of Face Recognition using The Viola-Jones Algorithm," 2021 International Conference on Artificial Intelligence and Mechatronics Systems (AIMS), Bandung, Indonesia, 2021, pp. 1-6, doi: 10.1109/AIMS52415.2021.9466010. Available at:
<https://ieeexplore.ieee.org/document/9466010>

jupyter Face_Detection Last Checkpoint: 08/09/2023 (autosaved)  [Logout](#)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Import library

```
In [1]: import os
import cv2
import dlib
import numpy as np
```

Predetermined variable

```
In [2]: # pathway to the dataset downloaded from kaggle
imageDir = 'raw_Dataset'

# determine the size to export image
size = (224,224)

# count number of processed images
empty,count,multiple = 0,0,0
```

Load the face detector

```
In [3]: detector = dlib.get_frontal_face_detector()
```

Iterate through the file, detect faces from image, extract face features and export it

```
In [4]: # iterate through each folder in the directory
for root, _, files in os.walk(imageDir):
    # iterate through each file in the folder
    for file in files:
        # Label of each picture which is the name of folder
        label = os.path.basename(root)
        # joining folder path and file name to access each image
        imagePath = os.path.join(root,file)
        # avoid .ipynb_checkpoints file
        if file.startswith('.'):
            continue
        # get pixel value of each image
        img = cv2.imread(imagePath)
        # convert image into grayscale images
        gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # detect face from the image
        faces = detector(gray_image)

        if len(faces) == 0:
            print('No faces detected at      ', root.split('\\')[-1], " ", file)
            empty += 1
            continue

        # iterate through the face detected
        for face in faces:
            # only 1 face should be detected
            if len(faces) == 1:
                # read the position of face
                x, y, x1, y1 = face.left(), face.top(), face.right(), face.bottom()
                x,y = abs(x),abs(y)
                # store the position of face detected
                face = img[y:y1, x:x1]
                # resize the detected face into desire size
                resize_face = cv2.resize(face,size)

                # create new folder for the processed image
                newpath = 'preprocessed_Dataset/' + label + '/'
                # new path does not exist
                if not os.path.exists(newpath):
                    # create new path
                    os.makedirs(newpath)
                # store preprocessed image into new path
                cv2.imwrite(newpath + file,resize_face)
                count += 1
            # mutiple faces are detected hence skip this image

        else:
            print("Mutiple faces are detected at " + root.split('\\')[-1] + " " + file)
            multiple += 1
            break
```

```

No faces detected at Angelina Jolie 004_f61e7d0c.jpg
Multiple faces are detected at Angelina Jolie 006_9135205d.jpg
Multiple faces are detected at Angelina Jolie 010_f99d79e3.jpg
No faces detected at Angelina Jolie 055_aaaf063c.jpg
Multiple faces are detected at Angelina Jolie 076_1d914d5b.jpg
Multiple faces are detected at Angelina Jolie 080_e998ab00.jpg
No faces detected at Angelina Jolie 090_da55509f.jpg
No faces detected at Brad Pitt 027_78f200c3.jpg
No faces detected at Brad Pitt 028_181cbb8a.jpg
No faces detected at Brad Pitt 054_9f01aefa.jpg
No faces detected at Brad Pitt 060_136e5ef5.jpg
Multiple faces are detected at Brad Pitt 064_213e2850.jpg
No faces detected at Brad Pitt 067_571d88eb.jpg
Multiple faces are detected at Brad Pitt 084_4876da64.jpg
No faces detected at Denzel Washington 006_2880115c.jpg
Multiple faces are detected at Denzel Washington 015_72bb2861.jpg
No faces detected at Denzel Washington 059_3b848154.jpg
No faces detected at Denzel Washington 093_4027b1e2.jpg
Multiple faces are detected at Hugh Jackman 044_1844df0f.jpg
Multiple faces are detected at Hugh Jackman 064_12d52b76.jpg
No faces detected at Hugh Jackman 095_b464eda0.jpg
No faces detected at Johnny Depp 018_14e5366f.jpg
No faces detected at Johnny Depp 043_77e393de.jpg
No faces detected at Johnny Depp 053_5b42d9d9.jpg
No faces detected at Johnny Depp 091_c3ad83af.jpg
No faces detected at Kate Winslet 047_c6dd0c78.jpg
No faces detected at Kate Winslet 050_0c20b215.jpg
No faces detected at Kate Winslet 082_4bbff9c3.jpg
Multiple faces are detected at Leonardo DiCaprio 073_42e32f65.jpg
No faces detected at Megan Fox 016_bd51d057.jpg
No faces detected at Megan Fox 023_55dd20e3.jpg
No faces detected at Natalie Portman 038_4930b73f.jpg
No faces detected at Natalie Portman 059_c13d7734.jpg
No faces detected at Natalie Portman 061_029fc37f.jpg
No faces detected at Natalie Portman 082_6883328f.jpg
No faces detected at Nicole Kidman 016_a669f24c.jpg
No faces detected at Nicole Kidman 040_dec66c8a.jpg
No faces detected at Nicole Kidman 076_a3e008c1.jpg
Multiple faces are detected at Robert Downey Jr 023_51dce41c.jpg
Multiple faces are detected at Robert Downey Jr 057_8572457a.jpg
Multiple faces are detected at Robert Downey Jr 061_b36635a2.jpg
No faces detected at Robert Downey Jr 074_a86aab43.jpg
No faces detected at Robert Downey Jr 076_318ed434.jpg
Multiple faces are detected at Sandra Bullock 023_7214d0b2.jpg
No faces detected at Sandra Bullock 026_e5342024.jpg
No faces detected at Sandra Bullock 032_f3773aa0.jpg
Multiple faces are detected at Sandra Bullock 035_bc2e1105.jpg
No faces detected at Sandra Bullock 043_4f352240.jpg
No faces detected at Sandra Bullock 044_49497c96.jpg
Multiple faces are detected at Sandra Bullock 071_45baaf8f.jpg
No faces detected at Sandra Bullock 078_d9a9ca25.jpg
No faces detected at Sandra Bullock 098_529825da.jpg
No faces detected at Sandra Bullock 100_e1433988.jpg
No faces detected at Scarlett Johansson 002_ea6e259d.jpg
No faces detected at Scarlett Johansson 029_75cdeb2c.jpg
No faces detected at Scarlett Johansson 037_ec9df32c.jpg
No faces detected at Scarlett Johansson 041_9913ca04.jpg
No faces detected at Scarlett Johansson 050_c8461888.jpg
No faces detected at Scarlett Johansson 077_776d5e0f.jpg
Multiple faces are detected at Scarlett Johansson 087_bb9186aa.jpg
No faces detected at Scarlett Johansson 102_97fca716.jpg
No faces detected at Scarlett Johansson 127_9894295d.jpg
No faces detected at Scarlett Johansson 128_40ed3346.jpg
No faces detected at Scarlett Johansson 137_29b565f9.jpg
No faces detected at Scarlett Johansson 142_5d78bc85.jpg
No faces detected at Scarlett Johansson 152_a5f9d1a6.jpg
No faces detected at Scarlett Johansson 182_56820995.jpg
Multiple faces are detected at Scarlett Johansson 195_dc745cfd.jpg
No faces detected at Tom Cruise 033_d891acbd.jpg
No faces detected at Tom Cruise 062_d4b7903f.jpg
No faces detected at Tom Cruise 078_076c27b8.jpg
No faces detected at Tom Cruise 080_566ea9e9.jpg
No faces detected at Tom Hanks 022_df2ce089.jpg
No faces detected at Tom Hanks 059_c7c906d9.jpg
No faces detected at Tom Hanks 067_15f0e6bb.jpg
No faces detected at Tom Hanks 100_b712e7ca.jpg
Multiple faces are detected at Will Smith 026_4f5bf02c.jpg
No faces detected at Will Smith 036_8751d89b.jpg
No faces detected at Will Smith 055_f9cbb53e.jpg
No faces detected at Will Smith 063_46f560ca.jpg
No faces detected at Will Smith 065_5cb55293.jpg
No faces detected at Will Smith 075_65ffca63.jpg
No faces detected at Will Smith 083_a0692bc1.jpg
No faces detected at Will Smith 093_dc555290.jpg
No faces detected at Will Smith 099_d652e3b6.jpg

```

```


In [5]: print('There are', empty, 'faces not being detected')
        print('There are', count, 'faces being detected and saved')
        print('There are', multiple, 'images detected multiple faces')

```

```

There are 66 faces not being detected
There are 1715 faces being detected and saved
There are 19 images detected multiple faces

```

jupyter Siamese_Model Last Checkpoint: 09/01/2023 (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Import libraries

```
In [1]: import os
import cv2
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import cycle
import keras
import pickle

import tensorflow as tf
from tensorflow.keras import layers
import tensorflow.keras.backend as K
from tensorflow.keras.layers import Input
from tensorflow.keras.applications import resnet
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import label_binarize
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
```

```
In [2]: import warnings
warnings.filterwarnings("ignore")
```

Set GPU Growth

```
In [3]: # Avoid OOM errors by setting GPU Memory Consumption Growth
physical_devices = tf.config.list_physical_devices('GPU')
for device in physical_devices:
    tf.config.experimental.set_memory_growth(device, True)
```

Data Preprocessing

```
In [4]: def load_data(ImageDir, augmentation=False):
    images, labels = [], []

    # Create an ImageDataGenerator instance with augmentation configurations
    if augmentation:
        datagen = ImageDataGenerator(
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
        )

    # iterate through each celebrity
    for celebs in os.listdir(ImageDir):
        # iterate through each file in features
        for file in os.listdir(os.path.join(ImageDir, celebs)):
            img = cv2.imread(os.path.join(ImageDir, celebs, file))

            if augmentation:
                x = img.reshape((1,) + img.shape)
                i = 0
                for batch in datagen.flow(x, batch_size=1):
                    augmented_img = batch[0].astype(np.uint8)
                    images.append(augmented_img)
                    labels.append(celebs)
                    i += 1
                    if i >= 5: # Generate 5 augmented images per original image
                        break
            else:
                images.append(img)
                labels.append(celebs)

    return images, labels
```

```
In [5]: images, labels = load_data('../Facial dataset/Preprocessed dataset', augmentation=False)
le = LabelEncoder()
numlabels = le.fit_transform(labels)
```



```
In [6]: def make_triplets(images,labels):
        A,P,N = [],[],[]
        numClasses = len(np.unique(labels))
        # index of labels for each class
        idx = [np.where(labels == i)[0] for i in np.arange(0, numClasses)]
        for i in np.arange(len(images)):
            # store current images
            A.append(images[i])
            label = labels[i]
            # positive images (same class with anchor)
            P_index = np.random.choice(idx[label])
            P.append(images[P_index])
            # negative images (different class with anchor)
            N_index = np.random.choice(np.where(labels != label)[0])
            N.append(images[N_index])

        A = np.array(A)
        P = np.array(P)
        N = np.array(N)

        return A,P,N
```

Split data and create triplets

```
In [7]: X_train,X_temp,y_train,y_temp = train_test_split(images,numlabels,test_size=0.3,random_state=42)
        X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```
In [8]: train_anchor,train_positive,train_negative = make_triplets(X_train,y_train)
        val_anchor,val_positive,val_negative = make_triplets(X_val,y_val)
        test_anchor,test_positive,test_negative = make_triplets(X_test,y_test)
```

```
In [9]: def display_triplets(A,P,N,seed=0):
        # randomize seed if not specified
        if seed == 0:
            seed = np.random.randint(0,len(A))
        else:
            seed = seed
        plt.subplot(1,3,1),plt.imshow(cv2.cvtColor(A[seed], cv2.COLOR_BGR2RGB)),plt.title('Anchor')
        plt.subplot(1,3,2),plt.imshow(cv2.cvtColor(P[seed], cv2.COLOR_BGR2RGB)),plt.title("Positive")
        plt.subplot(1,3,3),plt.imshow(cv2.cvtColor(N[seed], cv2.COLOR_BGR2RGB)),plt.title('Negative')
        plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[]);
        plt.show()
```

```
In [10]: print("Training triplets")
        display_triplets(train_anchor,train_positive,train_negative)
        print("Validation triplets")
        display_triplets(val_anchor,val_positive,val_negative)
        print("Testing triplets")
        display_triplets(test_anchor,test_positive,test_negative)
```

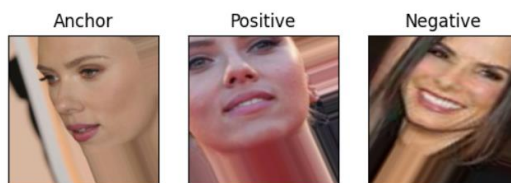
Training triplets



Validation triplets



Testing triplets



Model Architecture

```
In [12]: def create_embedding_model(input_shape):
# construct the input layer and pass the inputs through a
# pre-processing layer
inputs = keras.Input(input_shape)
x = resnet.preprocess_input(inputs)

# fetch the pre-trained resnet 50 model and freeze the weights
baseCnn = resnet.ResNet50(weights="imagenet", include_top=False)
baseCnn.trainable=False

# pass the pre-processed inputs through the base cnn and get the
# extracted features from the inputs
extractedFeatures = baseCnn(x)
# pass the extracted features through a number of trainable layers
x = layers.GlobalAveragePooling2D()(extractedFeatures)
x = layers.Dense(units=1024, activation="relu")(x)
x = layers.Dropout(0.5)(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(units=512, activation="relu")(x)
x = layers.Dropout(0.5)(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(units=256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(units=128)(x)
# build the embedding model and return it
embedding = keras.Model(inputs, outputs, name="embedding")
return embedding
```

```
In [11]: stops
```

```
In [13]: feature_extractor = create_embedding_model((224,224,3))
feature_extractor.summary()
```

Model: "embedding"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
tf.__operators__.getitem (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add (TFOpLambda)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, None, None, 2048)	23587712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
batch_normalization (Batch Normalization)	(None, 1024)	4096
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
batch_normalization_1 (Batch Normalization)	(None, 512)	2048
dense_2 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
Total params: 26381056 (100.64 MB)		
Trainable params: 2790272 (10.64 MB)		
Non-trainable params: 23590784 (89.99 MB)		

```
In [14]: def create_siamese_network(input_shape,embedding_model):
# Define tensors for the triplet of input images
anchor_input = Input(input_shape,name='anchor_input')
positive_input = Input(input_shape,name='positive_input')
negative_input = Input(input_shape,name='negative_input')

# Generate embedding outputs
encoded_anchor = embedding_model(anchor_input)
encoded_positive = embedding_model(positive_input)
encoded_negative = embedding_model(negative_input)

inputs = [anchor_input,positive_input,negative_input]
outputs = [encoded_anchor,encoded_positive,encoded_negative]

model = keras.Model(inputs=inputs,outputs=outputs,name='embedding')
return model
```

Create and Train Siamese model

```
In [15]: class SiameseModel(keras.Model):
def __init__(self, siameseNetwork, margin, lossTracker):
    super().__init__()
    self.siameseNetwork = siameseNetwork
    self.margin = margin
    self.lossTracker = lossTracker
def _compute_distance(self, inputs):
    (anchor, positive, negative) = inputs
    # embed the images using the siamese network
    embeddings = self.siameseNetwork((anchor, positive, negative))
    anchorEmbedding = embeddings[0]
    positiveEmbedding = embeddings[1]
    negativeEmbedding = embeddings[2]
    # calculate the anchor to positive and negative distance
    apDistance = tf.reduce_sum(
        tf.square(anchorEmbedding - positiveEmbedding), axis=-1
    )
    anDistance = tf.reduce_sum(
        tf.square(anchorEmbedding - negativeEmbedding), axis=-1
    )

    # return the distances
    return (apDistance, anDistance)
def _compute_loss(self, apDistance, anDistance):
    loss = apDistance - anDistance
    loss = tf.maximum(loss + self.margin, 0.0)
    return loss

def call(self, inputs):
    # compute the distance between the anchor and positive,
    # negative images
    (apDistance, anDistance) = self._compute_distance(inputs)

    # Get the embeddings from the siamese network
    (anchorEmbedding, positiveEmbedding, negativeEmbedding) = self.siameseNetwork(inputs)

    # Return the distances and embeddings
    return (apDistance, anDistance, anchorEmbedding, positiveEmbedding, negativeEmbedding)

def train_step(self, inputs):
    with tf.GradientTape() as tape:
        # compute the distance between the anchor and positive,
        # negative images
        (apDistance, anDistance) = self._compute_distance(inputs)
        # calculate the loss of the siamese network
        loss = self._compute_loss(apDistance, anDistance)
        # compute the gradients and optimize the model
        gradients = tape.gradient(
            loss,
            self.siameseNetwork.trainable_variables)
        self.optimizer.apply_gradients(
            zip(gradients, self.siameseNetwork.trainable_variables)
        )
        # update the metrics and return the loss
        self.lossTracker.update_state(loss)
        return {"loss": self.lossTracker.result()}
def test_step(self, inputs):
    # compute the distance between the anchor and positive,
    # negative images
    (apDistance, anDistance) = self._compute_distance(inputs)
    # calculate the loss of the siamese network
    loss = self._compute_loss(apDistance, anDistance)

    # update the metrics and return the loss
    self.lossTracker.update_state(loss)
    return {"loss": self.lossTracker.result()}
@property
def metrics(self):
    return [self.lossTracker]
```

Final data preparation

```

In [16]: # Create a generator function that yields the data in batches.
def data_generator(anchor, positive, negative, batch_size=32):
    while True:
        batch_indices = np.random.choice(len(anchor), batch_size, replace=False)
        anchor_batch = [anchor[i] for i in batch_indices]
        positive_batch = [positive[i] for i in batch_indices]
        negative_batch = [negative[i] for i in batch_indices]

        yield (np.array(anchor_batch), np.array(positive_batch), np.array(negative_batch))

In [17]: # Create tf.data.Dataset using the generator function.
batch_size = 32
train_dataset = tf.data.Dataset.from_generator(
    data_generator,
    args=(train_anchor, train_positive, train_negative, batch_size),
    output_signature=(
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
    )
)
val_dataset = tf.data.Dataset.from_generator(
    data_generator,
    args=(val_anchor, val_positive, val_negative, batch_size),
    output_signature=(
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(batch_size, 224, 224, 3), dtype=tf.float32),
    )
)

In [18]: # Create an EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitor validation loss
    patience=5, # Number of epochs with no improvement to wait
    restore_best_weights=True # Restore model weights to the best iteration
)

In [19]: input_shape = (224,224,3)
embedding_model = create_embedding_model(input_shape)
siameseNetwork = create_siamese_network(input_shape, embedding_model)
siameseModel = SiameseModel(
    siameseNetwork=siameseNetwork,
    margin=0.5,
    lossTracker=keras.metrics.Mean(name="loss")
)

In [20]: siameseModel.compile(optimizer=keras.optimizers.Adam(0.0001), weighted_metrics=[])
# Train the siamese model
history = siameseModel.fit(train_dataset, epochs=10, validation_data=val_dataset, validation_steps=10, steps_per_epoch=50, callbacks=[
    # Save the model
    # siameseModel.save("embedding_model", save_format='tf')
])

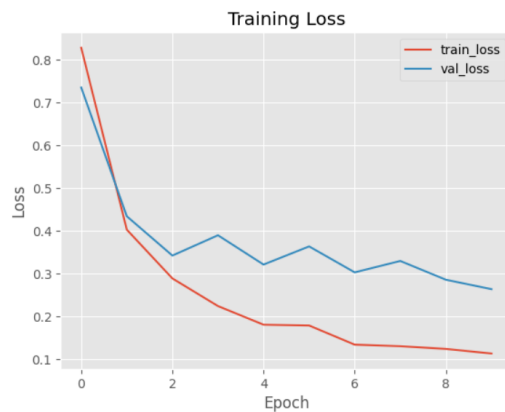
Epoch 1/10
50/50 [=====] - 781s 7s/step - loss: 0.8282 - val_loss: 0.7351
Epoch 2/10
50/50 [=====] - 333s 7s/step - loss: 0.4027 - val_loss: 0.4340
Epoch 3/10
50/50 [=====] - 334s 7s/step - loss: 0.2886 - val_loss: 0.3421
Epoch 4/10
50/50 [=====] - 335s 7s/step - loss: 0.2240 - val_loss: 0.3895
Epoch 5/10
50/50 [=====] - 334s 7s/step - loss: 0.1804 - val_loss: 0.3210
Epoch 6/10
50/50 [=====] - 333s 7s/step - loss: 0.1785 - val_loss: 0.3635
Epoch 7/10
50/50 [=====] - 342s 7s/step - loss: 0.1337 - val_loss: 0.3028
Epoch 8/10
50/50 [=====] - 334s 7s/step - loss: 0.1301 - val_loss: 0.3295
Epoch 9/10
50/50 [=====] - 336s 7s/step - loss: 0.1238 - val_loss: 0.2856
Epoch 10/10
50/50 [=====] - 336s 7s/step - loss: 0.1130 - val_loss: 0.2635

```

Training result

```
In [21]: def loss_plotting(H):
# construct a plot that plots and saves the training history
plt.style.use("ggplot")
plt.figure()
plt.plot(H.history["loss"], label="train_loss")
plt.plot(H.history["val_loss"], label="val_loss")
plt.title("Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.savefig('Loss_graph.png')
plt.show()
```

```
In [22]: # plot the loss training history
loss_plotting(history)
```



Feature Visualization

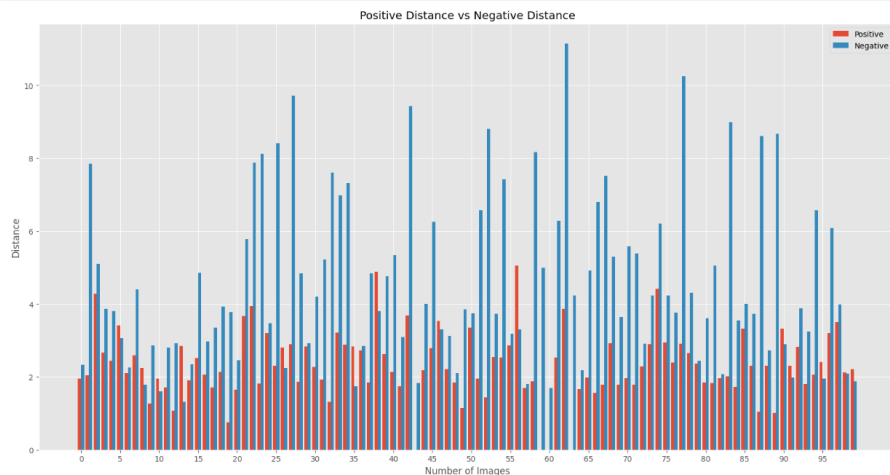
```
In [23]: train_pos_dist, train_neg_dist, trained_anchor, trained_positive, trained_negative = siameseModel.predict(
(train_anchor, train_positive, train_negative)
)
```

188/188 [=====] - 2217s 11s/step

```
In [24]: pos_dist, neg_dist, tested_anchor, tested_positive, tested_negative = siameseModel.predict(
(test_anchor, test_positive, test_negative)
)
```

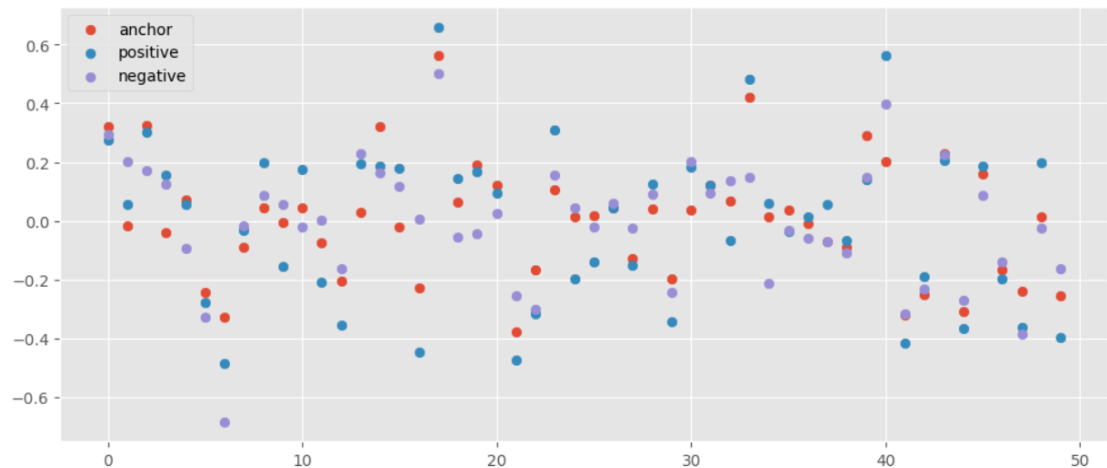
41/41 [=====] - 428s 10s/step

```
In [25]: range=100
X_axis = np.arange(range)
plt.figure(figsize=(20,10))
plt.bar(X_axis - 0.2, pos_dist[:range], 0.4, label = 'Positive')
plt.bar(X_axis + 0.2, neg_dist[:range], 0.4, label = 'Negative')
plt.xticks(np.arange(min(X_axis), max(X_axis)+1, 5.0))
plt.xlabel("Number of Images")
plt.ylabel("Distance")
plt.title("Positive Distance vs Negative Distance")
plt.legend()
plt.show()
```



```
In [26]: def display_features(A,P,N):
fig = plt.figure(figsize=(12,5))
ax1 = fig.add_subplot(111)
ax1.scatter(y=A[0][:50],x=np.arange(50),label='anchor')
ax1.scatter(y=P[0][:50],x=np.arange(50),label='positive')
ax1.scatter(y=N[0][:50],x=np.arange(50),label='negative')
plt.legend(loc='upper left')
plt.show()
```

```
In [27]: display_features(tested_anchor,tested_positive,tested_negative)
```



Classification model - KNN

```
In [28]: KNN_train = np.concatenate((trained_anchor,trained_positive))
KNN_test = np.concatenate((tested_anchor,tested_positive))
KNN_yTrain = np.concatenate((y_train,y_train.copy()))
KNN_yTest = np.concatenate((y_test,y_test.copy()))
```

```
In [29]: knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(KNN_train, KNN_yTrain)
y_pred = knn.predict(KNN_test)
print("Overall Accuracy:", accuracy_score(KNN_yTest, y_pred)*100,"%")
```

Overall Accuracy: 52.71950271950272 %

```
In [30]: # save the model to disk
filename = 'classification_model.sav'
pickle.dump(knn, open(filename, 'wb'))
```

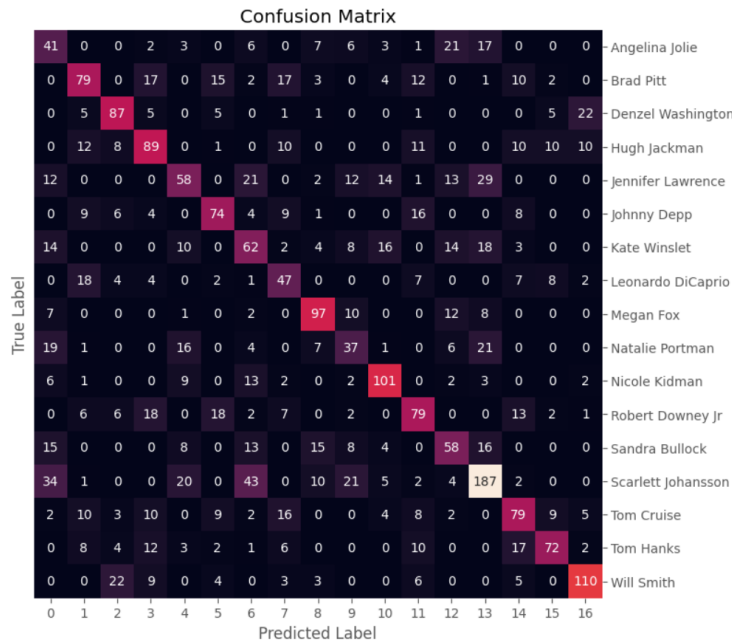
Evaluation

```
In [31]: labels_mapping = dict(zip(1e.classes_, 1e.transform(1e.classes_)))
sorted_labels = [label for label, _ in sorted(labels_mapping.items(), key=lambda x: x[1])]

fig, ax = plt.subplots(figsize=(10, 8))
mat = confusion_matrix(KNN_yTest,y_pred)
sns.heatmap(mat.T,square=True,annot=True,fmt='d',cbar=False)

# Set y-axis tick labels using the List of label names
ax.set_yticklabels(sorted_labels, rotation=0)
ax.yaxis.tick_right()

plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



```
In [32]: print(classification_report(KNN_yTest, y_pred))
```

	precision	recall	f1-score	support
0	0.38	0.27	0.32	150
1	0.49	0.53	0.51	150
2	0.66	0.62	0.64	140
3	0.55	0.52	0.54	170
4	0.36	0.45	0.40	128
5	0.56	0.57	0.57	130
6	0.41	0.35	0.38	176
7	0.47	0.39	0.43	120
8	0.71	0.65	0.68	150
9	0.33	0.35	0.34	106
10	0.72	0.66	0.69	152
11	0.51	0.51	0.51	154
12	0.42	0.44	0.43	132
13	0.57	0.62	0.59	300
14	0.50	0.51	0.50	154
15	0.53	0.67	0.59	108
16	0.68	0.71	0.70	154
accuracy			0.53	2574
macro avg	0.52	0.52	0.52	2574
weighted avg	0.53	0.53	0.53	2574

```
In [33]: # Binarize the Labels for ROC analysis
y_train_bin = label_binarize(KNN_yTrain, classes=np.unique(KNN_yTrain))
y_test_bin = label_binarize(KNN_yTest, classes=np.unique(KNN_yTrain))
n_classes = len(np.unique(KNN_yTrain))

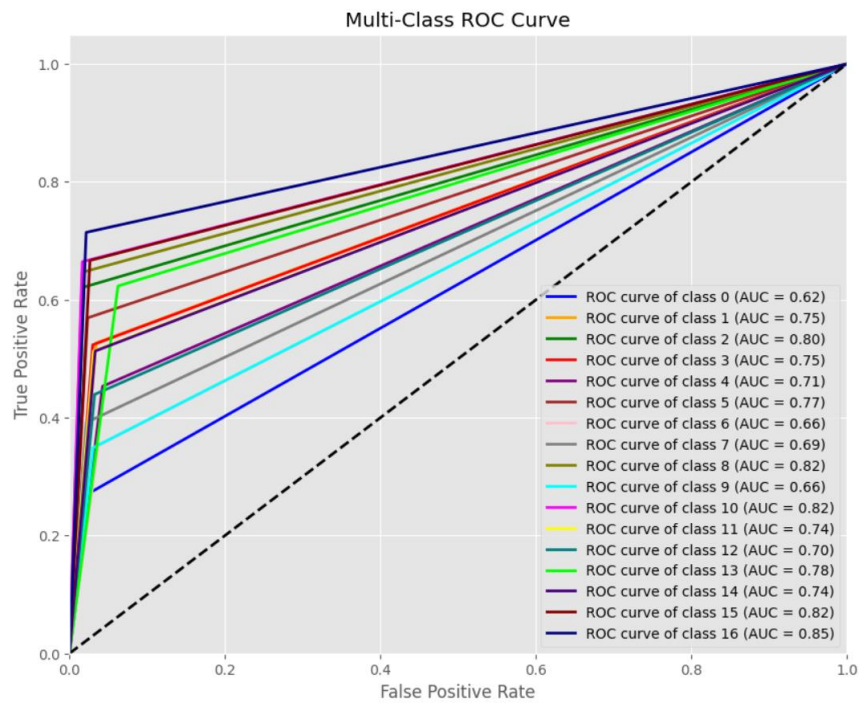
# Compute ROC curve and ROC AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in np.arange(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], knn.predict_proba(KNN_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
colors = cycle([
    'blue', 'orange', 'green', 'red', 'purple',
    'brown', 'pink', 'gray', 'olive', 'cyan',
    'magenta', 'yellow', 'teal', 'lime', 'indigo',
    'maroon', 'navy'
])

for i, color in zip(np.arange(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (AUC = {1:.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-Class ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

jupyter Face Recognition Pipeline Last Checkpoint: a minute ago (unsaved changes) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run Siamese model relevant function

```
In [1]: %run Siamese_Model.ipynb
```

Detect and Extract human faces

```
In [2]: %run Face_Detection.ipynb
```

```
There are 0 faces not being detected
There are 3 faces being detected and saved
There are 0 images detected multiple faces
```

Encode and load new unseen data

```
In [3]: new_le = LabelEncoder()
images, labels = load_data('preprocessed_Dataset', augmentation=True)
numlabels = new_le.fit_transform(labels)
```

Turn new data into triplets

```
In [4]: anchor, positive, negative = make_triplets(images, numlabels)
display_triplets(anchor, positive, negative)
```

Anchor

Positive

Negative

Embed triplets

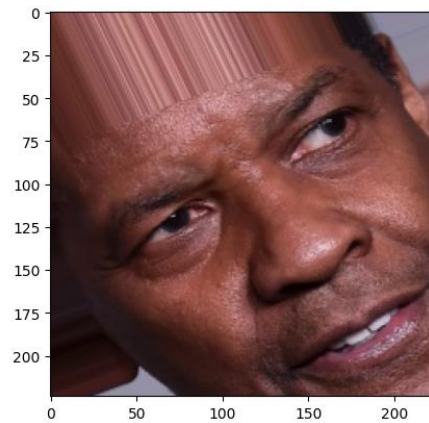
```
In [5]: # Load the model
model = keras.models.load_model("embedding_model", compile=False)
pos, neg, embedded_anchor, embedded_positive, embedded_negative = model.predict((anchor, positive, negative))

1/1 [=====] - 9s 9s/step
```

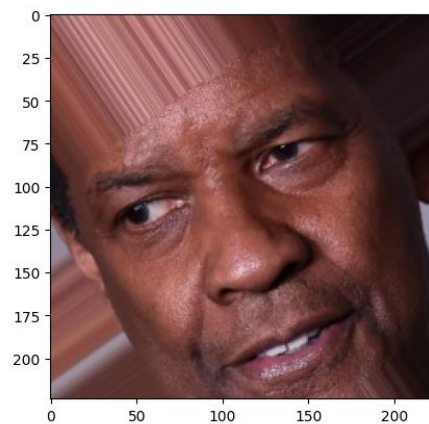
Classify embedded triplets and display

```
In [6]: filename = 'classification_model.sav'
knn = pickle.load(open(filename, 'rb'))
num_pred = knn.predict(np.concatenate((embedded_anchor, embedded_positive)))
name_pred = le.inverse_transform(num_pred)
for i in np.arange(embedded_anchor.shape[0]):
    print("Expected :", labels[i])
    print("Predicted:", name_pred[i])
    plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
    plt.show()
```

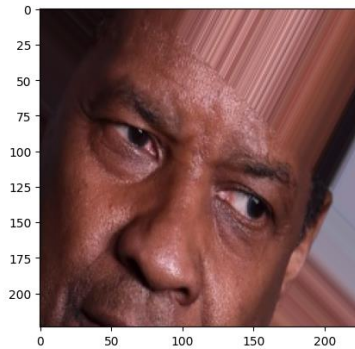
Expected : Denzel Washington
 Predicted: Denzel Washington



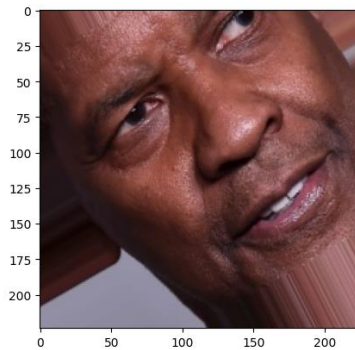
Expected : Denzel Washington
 Predicted: Denzel Washington



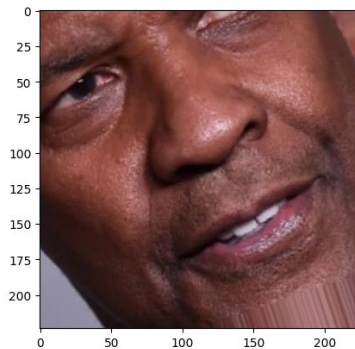
Expected : Denzel Washington
Predicted: Denzel Washington



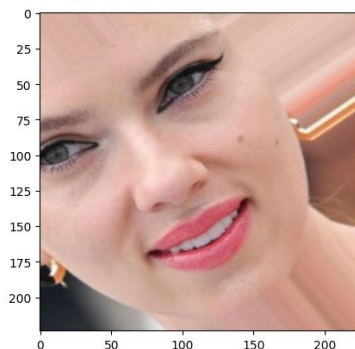
Expected : Denzel Washington
Predicted: Denzel Washington



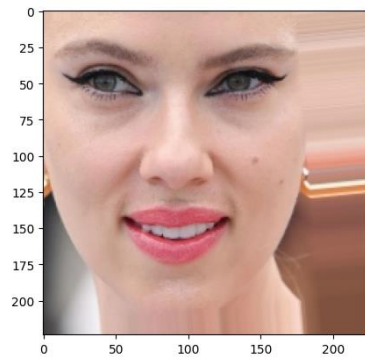
Expected : Denzel Washington
Predicted: Denzel Washington



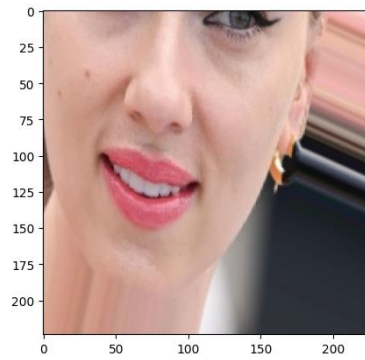
Expected : Scarlett Johansson
Predicted: Natalie Portman



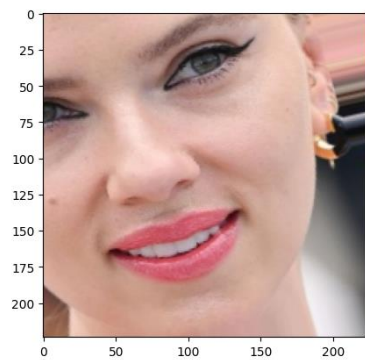
Expected : Scarlett Johansson
Predicted: Natalie Portman



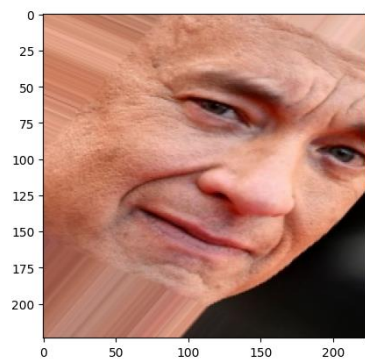
Expected : Scarlett Johansson
Predicted: Nicole Kidman



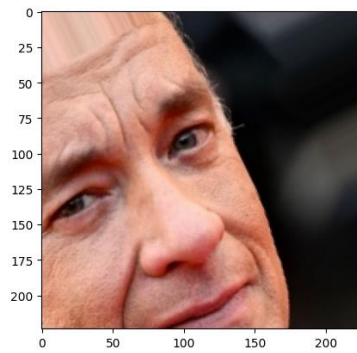
Expected : Scarlett Johansson
Predicted: Scarlett Johansson



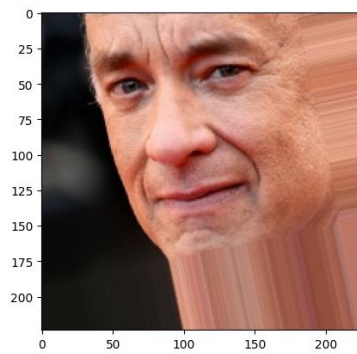
Expected : Tom Hanks
Predicted: Tom Hanks



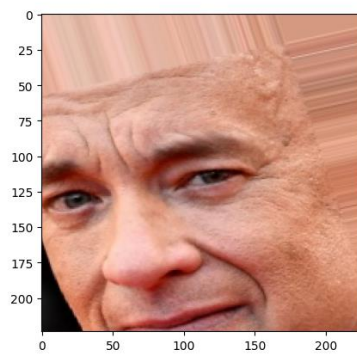
Expected : Tom Hanks
Predicted: Brad Pitt



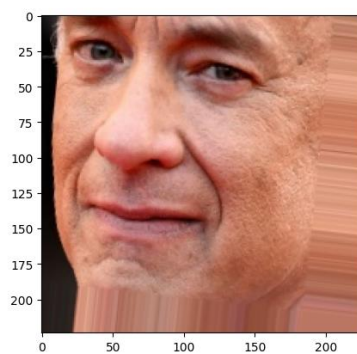
Expected : Tom Hanks
Predicted: Tom Hanks



Expected : Tom Hanks
Predicted: Tom Hanks



Expected : Tom Hanks
Predicted: Johnny Depp



```

In [7]: from plotly.subplots import make_subplots
import plotly.express as px
from sklearn.decomposition import PCA

# Assuming 'embedded_anchor', 'Labels', and 'create_embedding_model' are defined

# Calculate PCA embeddings for the first dataset
pca = PCA(n_components=2)
pca_embeddings = pca.fit_transform(embedded_anchor)

# Create the untrained model and calculate PCA embeddings for the second dataset
untrained_model = create_embedding_model((224,224,3))
untrained_anchor = untrained_model.predict(anchor)
pca = PCA(n_components=2)
untrained_embeddings = pca.fit_transform(untrained_anchor)

# Create subplots
fig = make_subplots(rows=1, cols=2, subplot_titles=("PCA Embeddings (Trained)", "PCA Embeddings (Untrained)"))

# Add scatter plots to subplots
scatter_trained = px.scatter(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], color=Labels)
scatter_untrained = px.scatter(x=untrained_embeddings[:, 0], y=untrained_embeddings[:, 1], color=Labels)

for trace in scatter_trained['data']:
    fig.add_trace(trace, row=1, col=1)

for trace in scatter_untrained['data']:
    fig.add_trace(trace, row=1, col=2)

# Update Layout
fig.update_layout(
    title="Embedding Space Visualization",
    xaxis_title="First Principal Component",
    yaxis_title="Second Principal Component",
    showlegend=False
)

fig.show()
1/1 [=====] - 2s 2s/step

```

Embedding Space Visualization

