Face Recognition using Siamese Network

A Project Report Submitted
In Partial Fulfillment of the Requirements
For the Award of the Degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Narendra Mohan Pathak: 210104033

Pratibha Singh: 210104039

Priyanshi Dhanuka: 210104040

Under the Supervision of

Dr. Rashi Agarwal

Associate Professor

Department of Computer Science and Engineering,

Harcourt Butler Technical University, Kanpur

To the
School of Engineering

Harcourt Butler Technical University, Kanpur

(Formerly Harcourt Butler Technological Institute, Kanpur)

Session: 2024-2025

Table of Contents

ABSTRACT

This project presents a comprehensive system for face recognition, integrating Convolutional Neural Networks and Siamese Networks within a FastAPI backend and a React frontend. Face recognition is a critical technology with applications ranging from security systems to personalized user experiences. Traditional methods often struggle with variations in pose, lighting conditions, and facial expressions. To address these challenges, this project proposes a novel approach that combines CNNs and Siamese networks to achieve robust and accurate face recognition.

The FastAPI backend acts as a bridge between the deep learning model and the frontend, providing an interface for communication. It handles incoming requests from the frontend, processes image data, and returns the results of face recognition tasks. The backend follows the RESTful architecture, which is designed for scalability and efficiency, ensuring seamless integration with the frontend.

The React frontend serves as the user interface, offering a user-friendly experience for interacting with the face recognition system. Users can upload images, initiate face recognition tasks, and view the results in real-time. The frontend communicates with the backend via API requests, enabling seamless interaction with the deep learning model.

Through extensive experimentation and evaluation, the system demonstrates its effectiveness in handling variations in pose, lighting conditions, and facial expressions.

CERTIFICATE

This is to certify that the project report titled "**Face Recognition using Siamese Network"** submitted by **'Narendra Mohan Pathak, Pratibha Singh, Priyanshi Dhanuka'** of Computer Science & Engineering Department, Harcourt Butler Technical University, in partial fulfilment of the requirements for the Bachelor of Technology has been completed under my guidance and supervision.

I certify that this project report is the result of their original work carried out by the group, and to the best of my knowledge, it does not contain any material previously published or written by another person, except where due reference is made.

This project report embodies the work carried out by the students during the final year of the Bachelor of Technology, and it meets the requirements and regulations governing the submission of such reports

Instructor's Name: Dr. Rashi Agarwal

Signature:

Date: 26/04/ 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who contributed to the completion of this project and supported us throughout its duration.

First and foremost, we extend our heartfelt appreciation to **Dr. Rashi Agarwal,** our project supervisor, for their invaluable guidance, encouragement, and constructive feedback. Their expertise and support were instrumental in shaping our ideas and refining our project.

We are grateful to the faculty members of the **Computer Science and Engineering Department** for their continuous support and for providing us with the necessary resources and facilities to carry out this project.

We also extend our thanks to our college **Harcourt Butler Technical University, Kanpur** for granting us permission to conduct our research and for providing access to essential data and information.

Our sincere thanks go to our classmates and friends for their encouragement, motivation, and assistance throughout the project. Their insights and discussions were invaluable in shaping our understanding and approach.

Narendra Mohan Pathak (210104033)
Pratibha Singh (210104039)
Priyanshi Dhanuka (210104040)

List of Figures

Chapter 1: Introduction

1.1 Problem Statement: Challenges in Face Recognition

Automatic face recognition plays a vital role in various security, access control, and human computer interaction applications. However, achieving accurate and reliable face recognition remains a challenge due to several factors:

- **Intra-class variations:** Human faces exhibit significant variations in pose, lighting conditions, facial expressions, aging, and even occlusions (e.g., glasses, masks). These variations can significantly hinder the recognition process.
- **Inter-class similarities:** Facial features can share similarities across different individuals, particularly for close relatives. This can lead to misidentification, especially with traditional recognition methods.
- **Limited training data:** Building robust face recognition models often requires large and diverse datasets. Collecting and labeling such datasets can be expensive and time consuming.

  These challenges can significantly impact the accuracy and reliability of face recognition systems.

1.2 Project Overview

Automatic face recognition has become an essential technology in various applications, including security systems, access control, and human-computer interaction. The ability to accurately recognize individuals from images and videos plays a critical role in these domains.

Traditional face recognition methods often struggle with variations in pose, lighting, and facial expressions. However, recent advancements in deep learning offer promising solutions. This paper explores the synergy between **Convolutional neural networks** and **Siamese networks** for robust face recognition.

1.3 Convolutional Neural Networks (CNNs):

Widely recognized for their success in image recognition, CNNs are adept at automatically extracting relevant features from visual data. In the context of face recognition, CNNs can effectively learn and represent key facial features such as eyes, nose, mouth, and their spatial relationships.

1.3.1 Key Terminologies of CNN:

Convolutional Layer:

- In a CNN, convolutional layers are responsible for extracting features from the input data through convolution operations. Each convolutional layer consists of multiple filters (also known as kernels) that slide across the input data, computing dot products between the filter weights and the input data to produce feature maps.

Filter/Kernel:

- A filter or kernel is a small matrix of weights that is convolved with the input data. During the convolution operation, the filter moves across the input data, computing dot products at each position. Filters learn to detect specific patterns or features in the input data, such as edges, textures, or shapes.

Stride:

- Stride refers to the step size at which the filter moves across the input data during the convolution operation. A larger stride value results in a smaller output feature map, while a smaller stride value leads to a larger output feature map.

Padding:

- Padding is the process of adding additional border pixels around the input data before applying convolutional operations. Padding helps preserve spatial information and can prevent loss of information at the edges of the input data. Common padding techniques include zero-padding and valid-padding.

Activation Function:

- An activation function introduces non-linearity to the output of a convolutional layer. Popular activation functions used in CNNs include ReLU (Rectified Linear Unit),sigmoid, and tanh. ReLU is widely used due to its computational efficiency and ability to alleviate the vanishing gradient problem.

Pooling Layer:

- Pooling layers are used to down sample the spatial dimensions of the feature maps produced by convolutional layers. Common pooling operations include max pooling and average pooling, which reduce the size of the feature maps while retaining the most relevant information.

Fully Connected Layer:

- Fully connected layers, also known as dense layers, are traditional neural network layers where each neuron is connected to every neuron in the previous layer. In CNNs, fully connected layers are typically used at the end of the network to perform classification or regression tasks based on the extracted features.

Dropout:

- Dropout is a regularization technique used to prevent overfitting in CNNs. During training, a fraction of neurons in a layer are randomly dropped out (set to zero) with a certain probability. Dropout forces the network to learn more robust features and reduces the reliance on individual neurons.

1.3.2 CNN Model Architecture

```
import numpy as np
import tensorflow as tf
import keras
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
import matplotlib.pyplot as plt
from keras.layers.normalization import BatchNormalization
from keras_preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_dir="../input/face-recognition-dataset/Original Images/Original Images/"
generator = ImageDataGenerator()
train_ds = generator.flow_from_directory(train_dir,target_size=(224, 224),batch_size=32)
classes = list(train_ds.class_indices.keys())
```

```
Found 2562 images belonging to 31 classes.
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(len(classes),activation='softmax'))
```

```
model.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'adam',
    metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 111, 111, 32)      0
_________________________________________________________________
batch_normalization_10 (Batc (None, 111, 111, 32)      128
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 54, 54, 64)        0
_________________________________________________________________
batch_normalization_11 (Batc (None, 54, 54, 64)        256
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 52, 52, 64)        36928
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 26, 26, 64)        0
_________________________________________________________________
batch_normalization_12 (Batc (None, 26, 26, 64)        256
```

```
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 10, 10, 32)        9248
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
batch_normalization_14 (Batc (None, 5, 5, 32)          128
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               102528
_________________________________________________________________
dense_5 (Dense)              (None, 31)                3999
=================================================================
Total params: 191,455
Trainable params: 191,007
Non-trainable params: 448
_________________________________________________________________
```

1.4 Siamese Networks:

These specialized architectures excel at learning similarity between inputs. They typically consist of two or more identical sub-networks that process separate inputs. The core lies in comparing the outputs of these sub-networks to determine the degree of similarity between the original inputs. For face recognition, Siamese networks compare a query face image to a known face representation stored in a database, assessing their similarity and enabling identification.

Siamese Network Model Code

```python
class DistanceLayer(layers.Layer):
    def __init__(self, p=2, **kwargs):
        super().__init__(**kwargs)
        self.p = p

    def call(self, anchor, positive, negative):
        ap_distance = tf.reduce_sum(tf.abs(anchor - positive) ** self.p, axis=-1)
        an_distance = tf.reduce_sum(tf.abs(anchor - negative) ** self.p, axis=-1)
        return (ap_distance, an_distance)


def get_siamese_network_2(input_shape=(128, 128, 3), p=2):
    encoder = get_encoder(input_shape)

    # Input Layers for the images
    anchor_input   = layers.Input(input_shape, name="Anchor_Input")
    positive_input = layers.Input(input_shape, name="Positive_Input")
    negative_input = layers.Input(input_shape, name="Negative_Input")

    ## Generate the encodings (feature vectors) for the images
    encoded_a = encoder(anchor_input)
    encoded_p = encoder(positive_input)
    encoded_n = encoder(negative_input)

    # A Layer to compute the distances using Minkowski distance
    distances = DistanceLayer(p)(
        encoder(anchor_input),
        encoder(positive_input),
```

```
    distances = DistanceLayer(p)(
        encoder(anchor_input),
        encoder(positive_input),
        encoder(negative_input)
    )

    # Creating the Siamese Network model
    siamese_network_2 = Model(
        inputs=[anchor_input, positive_input, negative_input],
        outputs=distances,
        name="Siamese_Network_2"
    )
    return siamese_network_2

siamese_network_2 = get_siamese_network_2(p=2)  # You can specify the value of p here
siamese_network_2.summary()
```

```
Model: "Siamese_Network_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Anchor_Input (InputLayer)       [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
Positive_Input (InputLayer)     [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
Negative_Input (InputLayer)     [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
Encode_Model (Sequential)       (None, 256)          22043944    Anchor_Input[0][0]
                                                                 Positive_Input[0][0]
                                                                 Negative_Input[0][0]
__________________________________________________________________________________________________
Encode_Model (Sequential)       (None, 256)          22043944    Anchor_Input[0][0]
                                                                 Positive_Input[0][0]
                                                                 Negative_Input[0][0]
__________________________________________________________________________________________________
distance_layer_2 (DistanceLayer ((None,), (None,))   0           Encode_Model[3][0]
                                                                 Encode_Model[4][0]
                                                                 Encode_Model[5][0]
==================================================================================================
Total params: 22,043,944
Trainable params: 9,583,800
Non-trainable params: 12,460,144
__________________________________________________________________________________________________
```

1.4.1 Key Terminologies of Siamese Network:

Siamese Network:

- A Siamese Network is a type of neural network architecture that consists of two identical subnetworks (twins) with shared weights and parameters. These subnetworks process two input samples (often pairs) in parallel and generate feature vectors, which are then compared to measure similarity or dissimilarity.

Feature Extraction:

- Feature extraction is the process of transforming raw input data into a compact representation that captures relevant information for a specific task. In Siamese architecture, each subnetwork performs feature extraction by processing input samples and generating feature vectors representing their characteristics.

Shared Weights:

- Shared weights refer to the parameters (weights and biases) of the neural network that are common between the twin subnetworks in a Siamese architecture. By sharing weights, the network learns a common representation for both input samples, facilitating comparison and similarity measurement.

Distance Metric:

- A distance metric is a mathematical function used to quantify the similarity or dissimilarity between two feature vectors. Common distance metrics include Euclidean distance, cosine similarity, and Manhattan distance. In Siamese networks, the choice of distance metric influences the similarity measurement between feature vectors.

Contrastive Loss:

- Contrastive loss is a loss function used in Siamese networks to train the model for similarity learning tasks. It penalizes the model based on the distance between similar pairs of samples (with a small distance) and dissimilar pairs of samples (with a large distance). The objective is to minimize the distance between similar pairs while maximizing the distance between dissimilar pairs.

Triplet Loss:

- Triplet loss is another loss function commonly used in Siamese networks for similarity learning tasks. It operates on triplets of samples: an anchor sample, a positive sample (similar to the anchor), and a negative sample (dissimilar to the anchor). Triplet loss aims to minimize the distance between the anchor and positive samples while simultaneously maximizing the distance between the anchor and negative samples.

Embedding Space:

- The embedding space is a high-dimensional space where input samples are mapped to their corresponding feature vectors by the Siamese network. In this space, similar samples are expected to have feature vectors that are close together, while dissimilar samples are expected to have feature vectors that are far apart.

1.5 Project Goals

This research proposes a novel approach that leverages the strengths of convolutional neural networks and Siamese networks to address these issues. CNNs excel at extracting robust and discriminative features from facial images, while Siamese networks are adept at learning similarity metrics between inputs.

By combining these techniques, we aim to develop a face recognition system that can:

1. Achieve high recognition accuracy even under challenging conditions with variations in pose, lighting, and expressions.
2. Effectively distinguish between similar-looking individuals, leading to improved identification accuracy.
3. Leverage the power of deep learning to potentially overcome limitations associated with smaller training datasets.

This research investigates how leveraging the strengths of both CNNs and Siamese networks can lead to significant improvements in face recognition performance. CNNs are employed to extract robust and discriminative facial features, while the Siamese network framework learns to distinguish between faces of the same individual and those of different individuals.

This combination offers several potential benefits:

- **Improved Robustness:** The approach aims to achieve superior recognition accuracy even under challenging conditions with variations in pose, lighting, or facial expressions.

- **Enhanced Feature Learning:** CNNs act as powerful feature extractors, capturing the underlying structure of faces in a way that is optimized for the subsequent Siamese network comparison.
- **Efficient Recognition:** The Siamese network architecture allows for efficient comparison between query faces and known representations, facilitating real-time applications.

This report delves into the detailed architecture of the proposed system, its training methodology, and evaluates its performance on benchmark datasets. We aim to demonstrate the effectiveness of this approach for achieving state-of-the-art face recognition capabilities.

Chapter 2: LITERATURE REVIEW

Automatic face recognition has witnessed significant advancements with the rise of deep learning techniques. This section reviews existing research on face recognition using convolutional neural networks and Siamese networks, highlighting their strengths and limitations. The automated facial recognition system is widely used to track attendance in business, schools, and colleges, among other places. It is faster and more effective than manual approaches, and it eliminates the need for manual work entirely. In a classroom setting, the **VioJones methodology** is used for face detection, the Linear Discriminant Analysis is used for recognition. A deep neural network model is used for predicting the image's facial key points. At this point, there are 20 key points plotted around the person's face. Another CNN is used to predict the person based on the calculated ratios and angles. Face biometrics are used to correctly recognize the person. To accomplish this, the K-nearest neighbour method is used to recognize the face.[1]

Particle Swarm Optimization (PSO), a metaheuristic optimization algorithm, is used to optimize the approach. When KNN and PSO are combined, they produce an efficient method of predicting faces. For feature extraction from images, the kernel discriminates analysis method is used, and machine learning methods such as SVM and KNN are used for classification. The Yale and ORL dataset are used for training and testing in this case. The ORL dataset is slightly more accurate than the Yale dataset.[3]

The color information in a facial image is extremely important during the prediction process. The color image is transformed into a multiple color space model, which is then transformed into eigen vectors and eigen values. These are fed into the nearest neighbor classifier, which classifies them. They are fed into the nearest neighbor classifier, which classifies them. There is a claim that the accuracy of this method outweighs all other Facial Recognition approaches.

E-voting has been introduced in Indian elections using facial recognition and fingerprint techniques.[7]

CNN is also important in face recognition, which is common in many implementations with this use case. The CNN model can be simplified further by combining the convolutional and sampling layers, resulting in a higher recognition rate. Image matching is an alternative technique that does not require the extraction of key points. Using neural network-based features vectors, the network simply matches the images. The Euclidean distance is used to calculate similarity in this case. The feature vector is a Siamese network created using CNN, which is learned with the labeled dataset. The loss function in this case is contrastive loss. A triplet loss approach is proposed for extracting the expressive deep feature by incorporating it into a Siamese network. The effectiveness of this approach has been demonstrated through theoretical analysis.[5]

The **MTCNN algorithm** is used to implement a surveillance system that can track a person's face using multiple cameras and from raw datasets. In the medical industry, the Facial Recognition system has its own set of benefits, such as retrieving a person's medical history, reducing the time spent in the reception area making manual appointments, and avoiding the use of RFID and printed record files. Furthermore, trying different and efficient algorithms will increase the speed and accuracy of the model so that one day no human intervention is required for its application in major industries such as medical and defense.[10]

A fundamental aspect of deploying AI model APIs is the choice of model format and serialization method. Researchers have investigated various serialization formats such as HDF5, ONNX, and TensorFlow's Saved Model format. Studies by Abadi et al. (2015) and Ronneberger et al. (2015) highlight the importance of selecting a format that balances interoperability, efficiency, and ease of integration into different deployment environments [22].

The selection of an appropriate **API development framework** plays a crucial role in the development and deployment of AI model APIs. FastAPI, a modern Python-based framework known for its performance and ease of use, has gained popularity in recent years.[23] Research by Montague et al. (2020) showcases the capabilities of FastAPI for

building scalable and efficient APIs, making it a compelling choice for deploying AI models.[24]

Testing AI model APIs is a critical aspect of the deployment process to ensure the reliability and performance of the deployed system. Researchers have explored various testing methodologies, including unit testing, integration testing, and performance testing. Studies by Niedermayer et al. (2018) and Ghotra et al. (2020) emphasize the importance of comprehensive testing to validate model functionality, handle edge cases, and assess performance under different conditions.[25]

2.1 Face Recognition with CNNs

CNNs have become the dominant approach for face recognition due to their ability to automatically learn discriminative features from facial images. Pioneering works like AlexNet and VGGFace demonstrated the effectiveness of CNNs in achieving high recognition accuracy on benchmark datasets. Subsequent research focused on improving CNN architectures for face recognition, exploring deeper networks and incorporating attention mechanisms to focus on crucial facial regions.

2.2 Siamese Networks for Face Verification

Siamese networks offer a powerful framework for learning similarity between facial representations. Works by Chopra introduced Siamese networks for face verification, achieving state-of-the-art performance on challenging datasets.Extensions like Triplet Loss further enhanced the ability of Siamese networks to distinguish between similar and dissimilar faces.

2.3 Accuracy in CNN & Siamese Networks

The Enduring Power of CNNs and Siamese Networks in Face Recognition: A Look Towards the Future. Face recognition technology has rapidly evolved in recent years, fueled by the advancements in deep learning, particularly Convolutional Neural Networks and Siamese Networks. These powerful architectures have demonstrably improved the

accuracy and robustness of face recognition systems, paving the way for exciting possibilities in the future.

This essay delves into the projected future scope of CNNs and Siamese networks in face recognition, exploring areas of improvement, potential applications, and emerging trends.

2.4 Refining Accuracy and Generalizability

One key area of focus lies in enhancing the accuracy and generalizability of face recognition systems. CNNs are at the forefront of this endeavor. Researchers are continuously developing more sophisticated CNN architectures capable of extracting even finer details from facial features. This meticulous feature extraction will lead to more precise recognition, especially in challenging scenarios. Poor lighting conditions, partial occlusions (masks, sunglasses), and variations in facial pose often pose difficulties for recognition systems. By leveraging advanced CNN architectures, future systems will be able to overcome these challenges and achieve exceptional accuracy.

Siamese networks, on the other hand, excel in verification tasks, determining whether two faces belong to the same person. Here, advancements lie in utilizing techniques like triplet loss and metric learning. Triplet loss functions train the network to learn a distance metric where similar faces have smaller distances in the embedding space compared to dissimilar faces. This refined distance metric will empower Siamese networks to perform verification tasks with even greater accuracy.

Overcoming Data Scarcity: A significant challenge in face recognition, particularly for specific domains, is the scarcity of data. Security systems with a limited number of authorized users often face this limitation. Siamese networks shine in such scenarios due to their ability to excel with limited datasets. By focusing on learning distance metrics, these networks can effectively handle situations where there are only a few images per person. This makes them ideal for applications where data collection might be restricted or impractical.

Privacy-Preserving Recognition: As face recognition technology becomes more pervasive, privacy concerns are paramount. Here, advancements in CNNs hold immense

promise. Techniques like federated learning offer a solution where training happens on decentralized devices, keeping user data private. In federated learning, models are trained on local devices holding individual user data. Instead of directly sharing the data, only the updated model parameters are exchanged between devices, ensuring user privacy remains protected. This approach paves the way for wider adoption of face recognition in privacy-sensitive applications, such as secure access control systems.

Real-Time Applications: The future of face recognition hinges on enabling real-time applications. This necessitates significant advancements in both CNN architectures and hardware capabilities. Optimizations in CNN architectures will lead to faster processing, allowing for real-time face recognition in various applications. Advancements in hardware, such as the development of specialized processors for deep learning tasks, will further accelerate processing speeds. This convergence of architectural improvements and hardware advancements will make real-time face recognition a reality, enabling applications like video surveillance and access control to operate with exceptional efficiency.

Merging Strengths: A Symbiotic Future: The future of face recognition holds promise for even greater advancements by combining the strengths of CNNs and Siamese networks. A potential approach involves using a Siamese network architecture with a pre-trained CNN for feature extraction. The Siamese network would then focus on learning the distance metric between the extracted features, leveraging the power of both architectures for improved recognition and efficiency.

Embracing Unsupervised Learning: Another exciting frontier lies in incorporating unsupervised learning techniques. Currently, face recognition systems rely heavily on labeled datasets. However, real-world scenarios often present variations not captured in these datasets. By incorporating unsupervised learning methods, future models can learn from unlabeled data, enhancing their ability to handle unseen variations and improving generalizability.

Explainable AI: Building Trust in the System: As face recognition technology becomes more integrated into society, ensuring trust and transparency is crucial. Explainable AI (XAI) techniques aim to make CNN-based recognition systems more transparent. By understanding how these systems arrive at their classifications, users can develop greater trust in the technology. XAI research will play a vital role in fostering wider acceptance and responsible use of face recognition systems.

2.5 Limitations and Open Challenges

Despite significant progress, challenges remain. Here are some challenges:

- **Data Requirements**: The performance of these networks heavily relies on the availability and quality of training data. Limited or biased datasets may lead to suboptimal recognition accuracy, especially for underrepresented demographics.
- **Computational Complexity:** CNNs and Siamese Networks often require significant computational resources for training and inference, hindering their deployment on resource-constrained devices or in real-time applications.
- **Robustness to Variability:** While CNNs can capture robust features, the combined system's robustness to variations in pose, illumination, and facial expressions may still be limited, leading to recognition failures in challenging conditions.
- **Generalization:** Ensuring that the trained model generalizes well to unseen faces and environments remains a challenge, especially when faced with domain shifts or adversarial attacks.
- **Privacy Concerns:** Face recognition technology raises significant privacy concerns, including the potential for misuse, surveillance, and unauthorized access to personal information, necessitating robust privacy-preserving techniques.
- **Ethical Considerations:** There are ethical considerations regarding the deployment of face recognition systems, including issues related to bias, fairness, and discrimination, which need to be carefully addressed to mitigate potential harm.

- **Adversarial Attacks:** CNNs are vulnerable to adversarial attacks, where imperceptible perturbations to input images can lead to misclassification. Adapting Siamese Networks to defend against such attacks remains an open challenge.

Chapter 3: WORK METHEDOLOGY

This chapter details the working process of the face recognition system using Convolutional Neural Networks and Siamese Networks. We'll delve into the software requirements, system modules, data flow, and various diagrams used to design and implement the system.

3.1 Tech Stack

This section outlines the software needed to implement the system. It should include:

- **Deep Learning Libraries:** Keras, PyTorch & TensorFlow are the deep learning libraries used for building the CNN and Siamese Network models.
- **Backend:** Python, FastAPI were used while developing the APIs.
- **Frontend:** React.js is used for creating the user interfaces and Axios library for interacting with the REST API.
- **Tools Used:** Kaggle, Jupyter Notebook, Git, GitHub, Visual Studio Code and Postman.

3.2 Project Modules

Here, you'll describe the various modules that make up the system. Each module should have a clear function:

- **Data Pre-processing Module:** This module prepares the facial images for training and testing. This involves resizing, normalization, and potentially additional steps like landmark detection for alignment.
- **CNN Feature Extraction Module:** CNNs consist of convolutional layers, pooling layers and fully connected layers, with the output of each layer being a function of the output of previous layers. In order for a CNN to be used in image classification, its output must be invariant to semantically-irrelevant changes.
- **Siamese Network Module:** Detail the architecture of the Siamese network used for comparing facial representations. Explain how it takes the CNN-extracted features as input and performs the similarity comparison.

- **Training Module:** This includes details on the training data, loss function, optimizer, and hyperparameter tuning.
- **Recognition Module:** This involves extracting features from a query image, comparing it with known faces in the database using the Siamese network, and making a final identification decision based on a predefined threshold.
- **Backend Module:** This involves exporting the model in H5 format and creating a FastAPI-based API in Python. After importing the model and dependencies, the API's core functionality is implemented in a "/predict" POST route, where it accepts image inputs and determines resemblance. Finally, the API is deployed, typically leveraging cloud services like AWS for scalability and reliability. This backend process ensures efficient model integration, robust API development, and seamless deployment, enabling organizations to deliver AI-powered solutions effectively.
- **Frontend Module:** The frontend module of the Face Similarity Detection AI is built using React.js, a popular JavaScript library for building user interfaces. React.js was chosen for its component-based architecture, which allows for the modular development of UI elements. Additionally, React's virtual DOM ensures efficient rendering and updates, providing a seamless user experience.

3.3 Deep Learning Model

3.3.1 Data Flow Diagram for Deep Learning Model

The DFD visually represents the flow of data through the system. It should illustrate:

- **Data Inputs:** This includes the raw facial images used for training and testing.
- **Data Processing:** Show how the preprocessing module prepares the images.
- **Feature Extraction:** Depict how the CNN extracts features from the preprocessed images.
- **Feature Encoding:** The extracted features undergo encoding to represent them in a suitable format for similarity comparison.

- **Similarity Comparison:** Utilizing a Siamese network architecture, the model compares the encoded features to measure the similarity between facial images.
- **Recognition Output:** The output of the model can be either the identified individual, if performing facial recognition, or a recognition score indicating the degree of similarity between the input images

3.3.2 Data Flow Diagram of Siamese Network

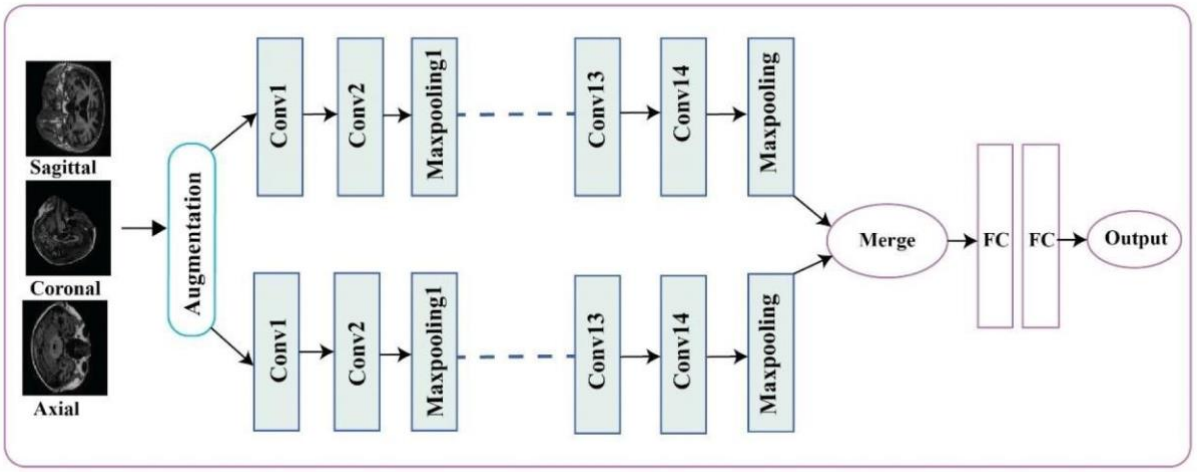


Fig. 3.1 DFD for Siamese Network

3.3.3 Data Flow Diagram of CNN

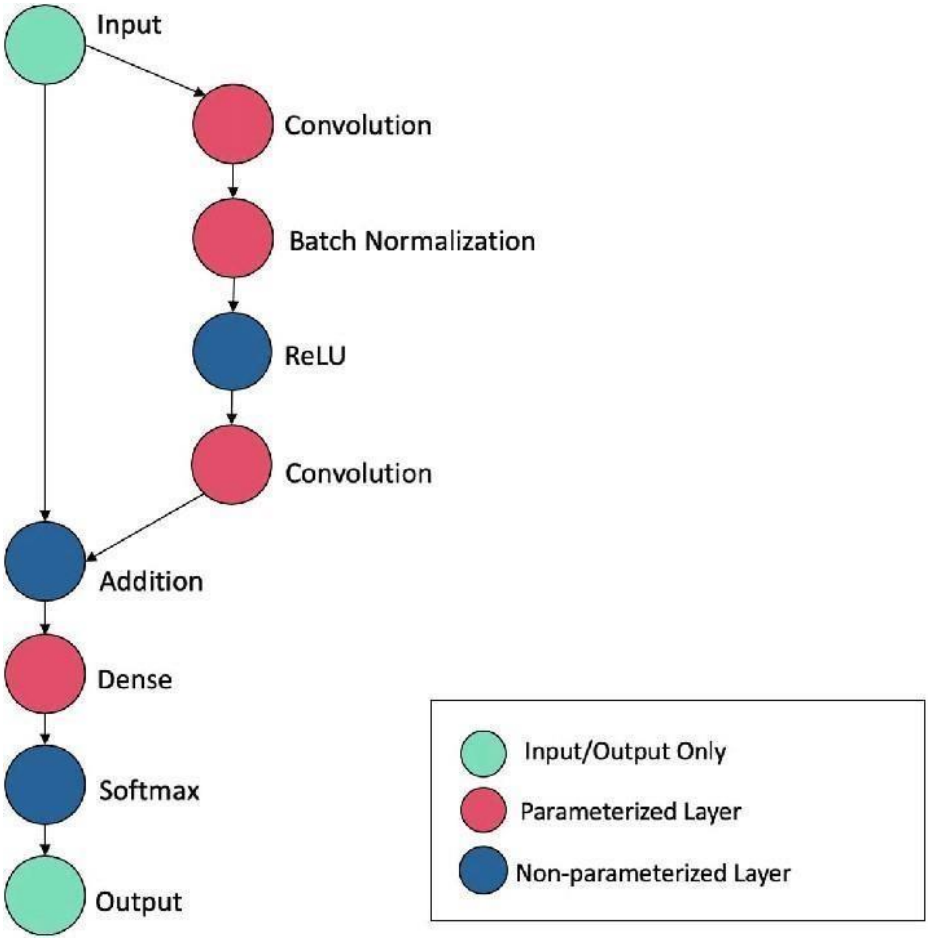


Fig. 3.2 DFD for CNN

3.3.4 Entity-Relationship Diagram

An ER diagram shows the relationships between entities in the system. This might be relevant if you have a database of known faces used for recognition. The diagram would show:

- **Entities:** This could be "Faces" and potentially additional entities like "Users" if user management is involved.
- **Attributes:** These are the properties of each entity, like "Image Path" and "Name" for a "Face" entity.
- **Relationships:** Show how entities relate to each other. For example, a "User" might have a "Many-to-Many" relationship with "Faces" if a user can have multiple known faces associated with them.

3.3.5 Use Case Diagram

A use case diagram depicts the interactions between actors (users) and the system. In this case, the actors could be:

- **System Administrator:** Responsible for setting up and training the system.
- **User:** Performs face recognition tasks using the trained system.
- **Actors:** Represent the system administrator and user.
- **Use Cases:** These are the functionalities the system offers, like "Train Face Recognition System" or "Recognize Face".
- **Interactions:** Show how actors interact with the system to perform these use cases.

3.3.6 Use Case Diagram of CNN

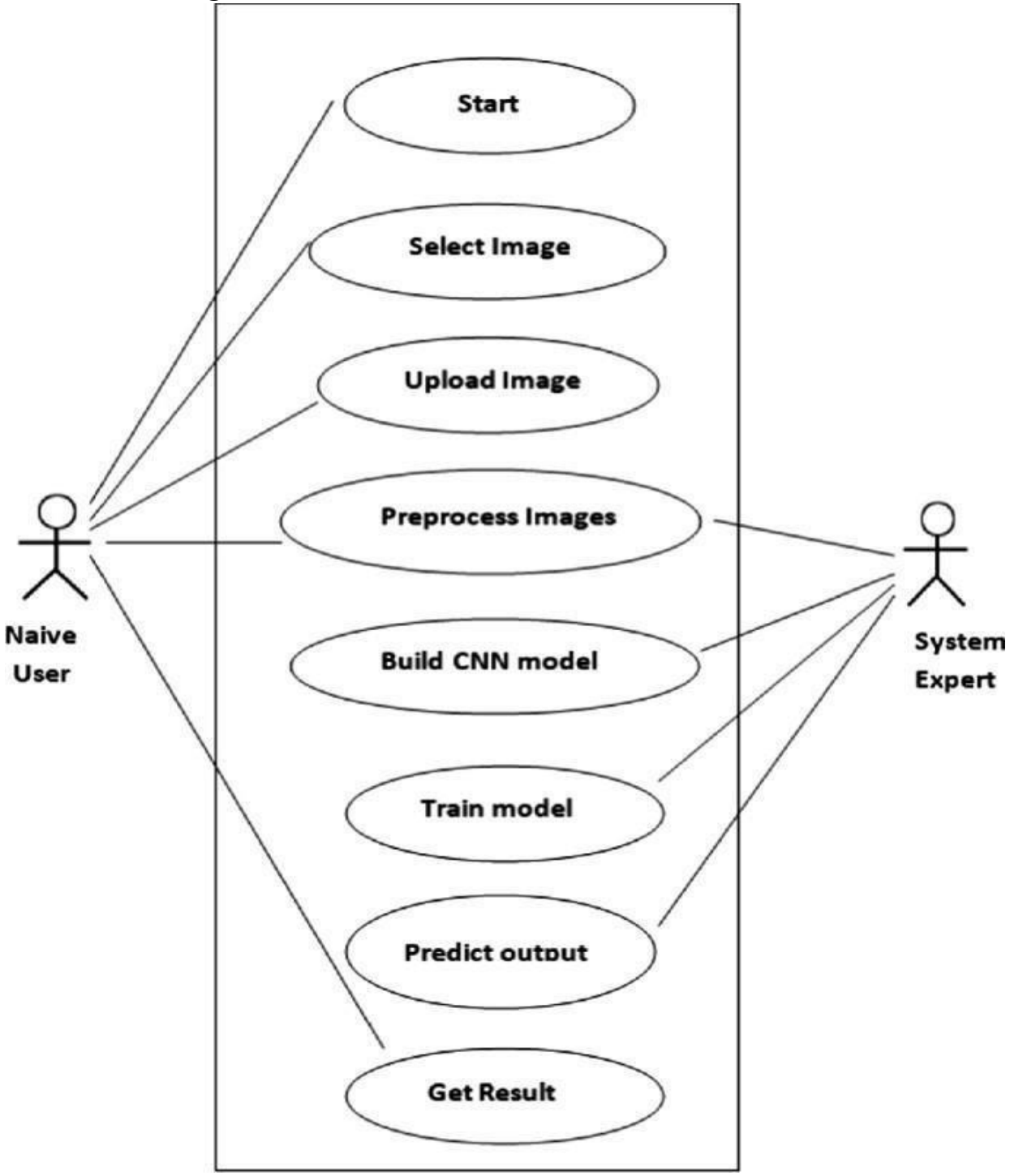


Fig. 3.3 Use Case Diagram for CNN Model

3.3.7 Architecture of CNN

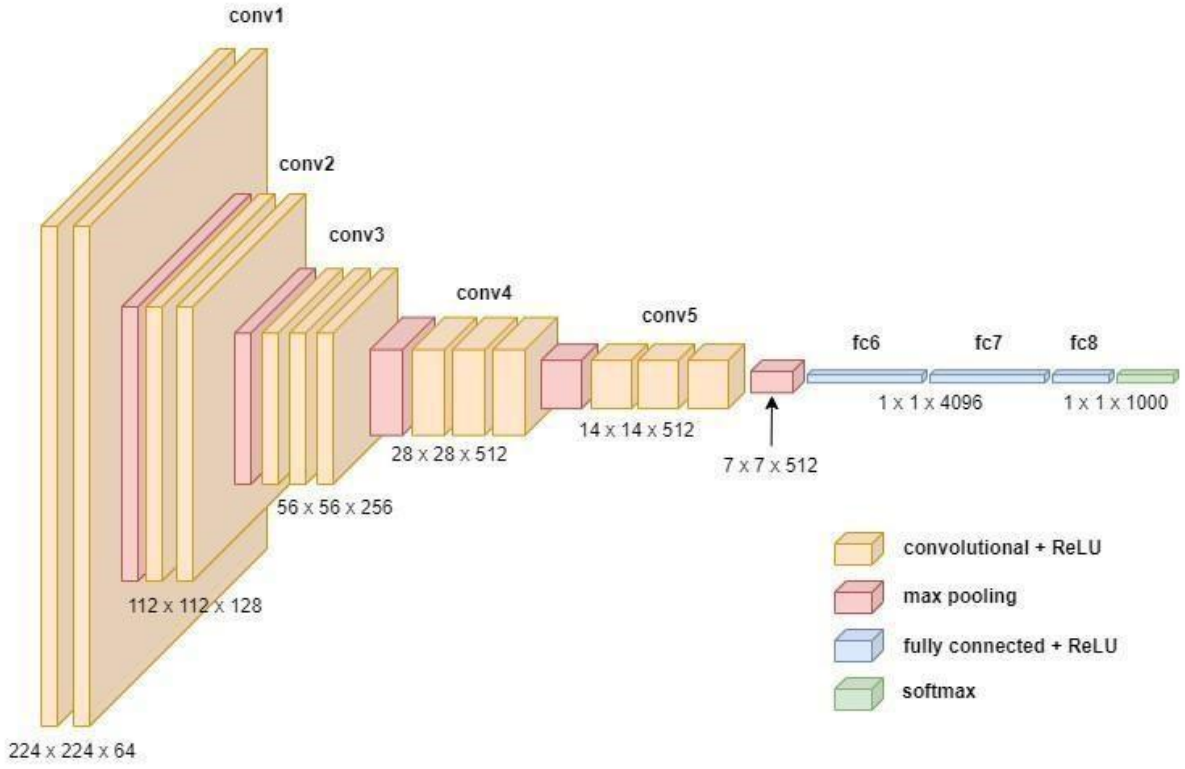


Fig. 3.4 Architecture of CNN

Architecture of the convolutional neural network model. ReLU is the rectified linear activation function. Spatial dropout is a 2D spatial dropout layer, dropping entire 2D feature maps. Batch norm is a batch normalization layer, normalizing the activation of the previous layer at each batch.

The Number of iterations (X) of the convolutional layers loop (red) is defined by the Adam optimization For every iteration, the number of filters of the loop is doubled. The first and potentially second convolutional layers loop contains a 2D max pooling layer of size **2** × **2**

The data are flattened (Flatten) after the convolutional layers loop. The number of iterations (Y) of the dense layers loop is defined by the Adam optimization. For every iteration, the number of units is doubled. The model is finalized with a dense layer with as

many units as there are different categories of windows (3 in this model) and a Softmax activation step.

3.3.8 Flow Diagram of Siamese Network

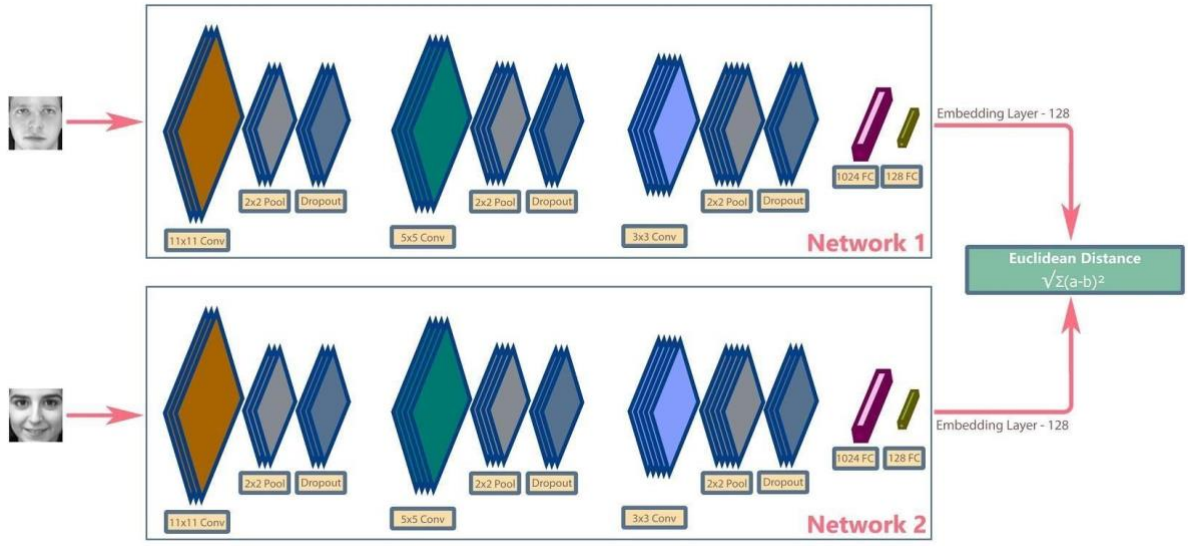


Fig. 3.5 Flow diagram of Siamese Network

3.3.9 Description of the Siamese Network

A Siamese Network is a type of neural network architecture that consists of two identical subnetworks (twins) with shared weights and parameters. These subnetworks process two input samples (often pairs) in parallel and generate feature vectors, which are then compared to measure similarity or dissimilarity.

Feature Extraction:

Feature extraction is the process of transforming raw input data into a compact representation that captures relevant information for a specific task. In Siamese architecture, each subnetwork performs feature extraction by processing input samples and generating feature vectors representing their characteristics.

Triplet Loss:

Triplet loss is another loss function commonly used in Siamese networks for similarity learning tasks. It operates on triplets of samples: an anchor sample, a positive sample (similar to the anchor), and a negative sample (dissimilar to the anchor). Triplet loss aims

to minimize the distance between the anchor and positive samples while simultaneously maximizing the distance between the anchor and negative samples.

3.4 Tools Used

The tools used in this project encompass a range of functionalities, from data exploration and model development to version control and API testing:

1. **Kaggle**: Kaggle serves as a platform for accessing datasets, participating in competitions, and collaborating with a global community of data scientists and machine learning practitioners. It provides a diverse range of datasets and kernels for exploring, analyzing, and building machine learning models.
2. **Jupyter Notebook:** Jupyter Notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It is widely used for interactive data analysis, prototyping machine learning models, and documenting research workflows.
3. **Git**: Git is a distributed version control system that enables collaboration, tracking changes, and managing code repositories. It allows developers to work on projects simultaneously, maintain version history, and merge changes seamlessly.
4. **GitHub**: GitHub is a web-based platform built on top of Git, providing hosting for software development projects. It offers features such as code hosting, pull requests, issue tracking, and collaboration tools, making it a popular choice for hosting opensource projects and facilitating team collaboration.
5. **Visual Studio Code**: Visual Studio Code (VS Code) is a lightweight yet powerful source code editor developed by Microsoft. It offers support for various programming languages, syntax highlighting, debugging, and extensions for customizing the development environment. It is commonly used for writing and debugging code, managing projects, and integrating with version control systems like Git.
6. **Postman**: Postman is a collaboration platform for API development that simplifies the process of building, testing, and documenting APIs. It provides a user-friendly interface for sending HTTP requests, testing API endpoints, and visualizing responses. Postman also offers features for automated testing, mock server creation.

Chapter 4: Design and Development Details

This chapter details the actual working process of the face recognition system using Convolutional Neural Networks and Siamese Networks

4.1 USER INTERFACE

4.1.1 Introduction

The Face Similarity Detection AI is a web application designed to compare the similarity between three facial images provided by the user. It serves as a tool for analyzing facial features and identifying similarities or differences between images. This report provides an overview of the application's functionality, implementation details, and future considerations.

4.1.2 Functionality

- **Image Upload:** Users can upload three facial images using the provided input fields. These images represent the ANCHOR image and two other image samples for comparison. Upon upload, the images are displayed in their respective drop zones.
- **Submit Button:** After uploading the images, users can submit them for analysis by clicking the "Submit" button. Upon submission, the client sends a POST request to the server. Then the server processes the images and calculates the similarity ratio between them.
- **Result Display:** Once the analysis is complete, the application displays the result in a dedicated section at the bottom of the page. The result includes a comparison between two images, along with their similarity ratio.

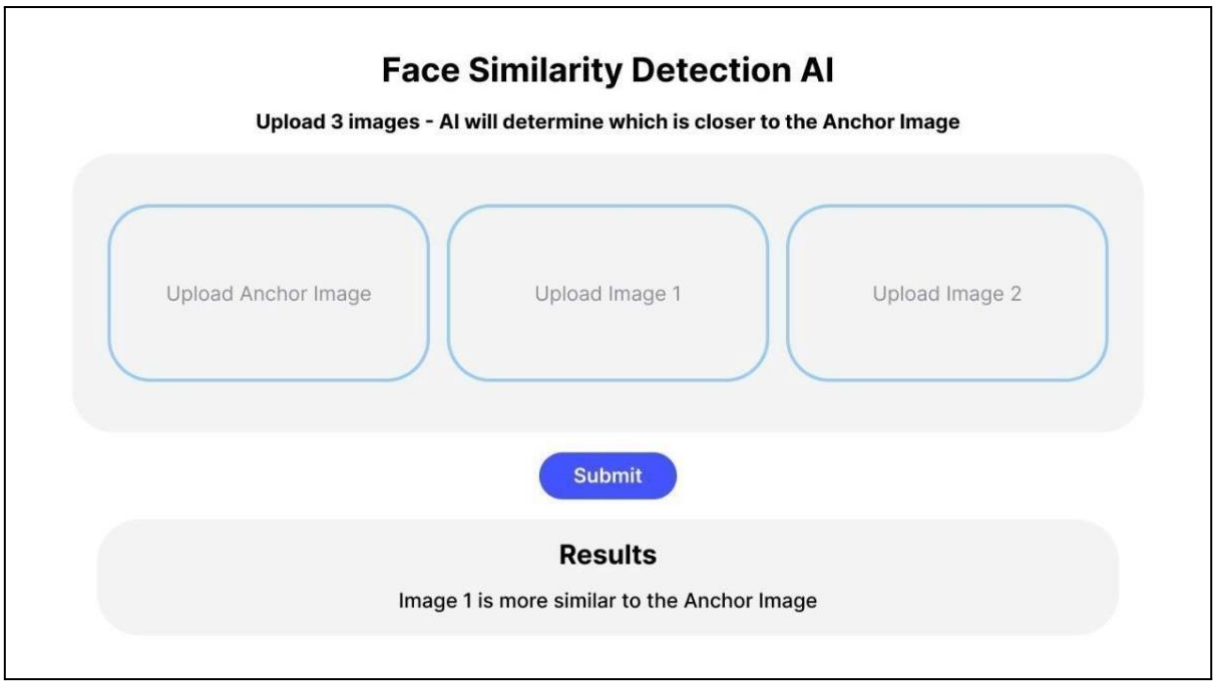


Fig. 4.1 UI design of the frontend application

4.1.3 Technologies Used

REACT

React is an open-source JavaScript library maintained by Facebook and a community of developers. It is used for building user interfaces, particularly single-page applications and complex web applications with dynamic content. React employs a component-based architecture, allowing developers to create reusable UI components that manage their own state and can be composed to build larger, more complex interfaces.

One of React's key features is its use of a virtual DOM (Document Object Model), which enables efficient updates to the UI by minimizing DOM manipulations. React uses a declarative syntax, where developers describe how the UI should look based on the application's state, rather than imperatively manipulating the DOM directly. React is often used in conjunction with other libraries and tools, such as Redux for state management, React

Router for handling routing in single-page applications, and webpack for bundling and managing dependencies. With its strong ecosystem, performance optimizations, and

developer-friendly approach, React has become one of the most popular choices for building modern web applications.

In the Face Similarity Detection AI, React is the core technology used for building the user interface and handling user interactions.

HTML and CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are fundamental web technologies used for structuring and styling web pages, respectively. HTML provides the structure and content of the web page, while CSS is used to define the layout, styling, and visual presentation of the elements. In the Face Similarity Detection AI, HTML and CSS are employed to create the layout of the application and style its components.

FileReader API

The FileReader API is a JavaScript API that provides asynchronous methods for reading the contents of files or raw data buffers. It allows web applications to read the contents of files selected by the user through file input elements. In the Face Similarity Detection AI, the FileReader API is utilized to read the contents of the uploaded image files and process them for further analysis.

Axios

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

One of the main advantages of Axios is its ease of use and flexibility. It supports all major HTTP methods (GET, POST, PUT, DELETE, etc.) and allows customization of request headers, timeout settings, and request/response interceptors. Axios also includes built-in support for handling request and response data in various formats, including JSON, FormData, and binary data. It automatically converts response data to JavaScript objects, simplifying data manipulation and error handling.

Another key feature of Axios is its support for Promise-based asynchronous programming. This allows developers to write clean and concise code using async/await syntax, making it easier to manage asynchronous operations and handle errors.

4.2 BACKEND

4.2.1 Introduction

In today's rapidly evolving technological landscape, integrating Artificial Intelligence (AI) models into real-world applications has become increasingly prevalent. As organizations strive to leverage the power of AI to enhance user experiences and drive business value, the need for efficient and scalable deployment solutions has become paramount. This report addresses the end-to-end process of developing and deploying an AI model API utilizing state-of-the-art technologies.

4.2.2 Steps involved

- Exporting the model in h5 format.
- Creating a starter API with FAST-API in Python.
- Importing the model & required dependencies.
- Making the "/predict" POST route, which accepts 3 images from the user.
- One image is an anchor and the other two are img1 and img2. The model will tell us which image resembles the anchor image.
- We return the result in JSON format to the client.

1. Exporting the model in h5 format

- This step involves saving the trained AI model in the Hierarchical Data Format (HDF5) format, commonly denoted by the `.h5` extension.

```
def load_final_model():
    model = load_model('./model/model_weights.h5')
    print("Model loaded")
    return model
```

- Model is imported via **load_model function** from tensorflow keras as shown above.

- HDF5 is a file format designed to store and organize large amounts of data efficiently. It's widely used in the machine learning community for saving models, datasets, and other structured data.
- When exporting the model to the h5 format, we can ensure that it can be easily loaded and utilized in other environments or frameworks.

2. Creating a starter API with FAST-API in Python

- FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.
- This step involves setting up a FastAPI project, which provides a foundation for building a web API that can serve your machine learning model.
- FastAPI's asynchronous capabilities make it particularly suitable for handling high loads and concurrent requests, which is beneficial for serving machine learning models in production environments.

3. Importing the model & required dependencies

- Before you can use the model in your API, you need to import it along with any necessary dependencies or libraries.
- This typically includes importing libraries. In our case we have Fastapi, tensorflow, numpy, Pillow, uvicorn as major dependencies.

4. Making the "/predict" POST route, which accepts 3 images from the user

- In this step, you define a POST endpoint `/predict` in your FastAPI application, which accepts three images as input from the client.
- These images consist of one anchor image and two comparison images (`img1` and `img2`).
- The below code shows the POST api route which accepts three images, pre-processes it and sends the data to the helper functions.

```
@app.post("/predict")
async def predict_api(fileAnchor: UploadFile = File(...), file1:
UploadFile = File(...), file2: UploadFile = File(...)):


    # extension check if not file_check(fileAnchor): return
    {"Error" : "Image must be jpg or png format!"} if not
    file_check(file1):



        return {"Error" : "Image must be jpg or png format!"} if
    not file_check(file2):
        return {"Error" : "Image must be jpg or png format!"}


    anchor = read_imagefile(await fileAnchor.read()) img1
    = read_imagefile(await file1.read()) img2
    = read_imagefile(await file2.read())


    try:
        prediction = predictSimilarImage(anchor, img1, img2)
    except Exception as e:
        return {"Error": str(e)}


    final_prediction = dict()


    if prediction:
              final_prediction.update({"Result" : "Image 1 is more
similar to Anchor Image"}) else:
               final_prediction.update({"Label" : "Image 2 is more
similar to Anchor Image"})



    return final_prediction
```

- The use of a POST request allows clients to send data (in this case, images) to the server for processing.

5. **The model will tell us which image resembles the anchor image**

- Once the server receives the images, the machine learning model is invoked to compare the similarity between the anchor image and the two comparison images.
- After post processing and error handling, API route calls a helper function **predictSimilarImage** which utlisies the model functions exported via h5 as shown in the below code.

```
def predictSimilarImage(anchor: Image.Image, img1: Image.Image,


img2: Image.Image):

    anchor = img_preprocess(anchor)


    img1 = img_preprocess(img1)


      img2 = img_preprocess(img2)



classifier = load_final_model()


     anchorWithImg1Test = classifier.classify_images(anchor, img1)
     anchorWithImg2Test = classifier.classify_images(anchor, img2)



if anchorWithImg1Test <= anchorWithImg2Test:

    return True else:


    return False
```

- The model computes a similarity score or makes a binary decision indicating which comparison image more closely resembles the anchor image based on the learned features.

6. Returning the result in JSON format to the client

- After the model has processed the input images and made its comparison, the result is formatted into a JSON (JavaScript Object Notation) response.

4.3.3 Testing the API.

To ensure the reliability and effectiveness of the deployed API, comprehensive testing is essential. This includes both unit testing and integration testing of various API endpoints, including the "/predict" route. Unit tests validate individual components of the API, such as input validation and model inference logic, while integration tests verify the functionality of the API as a whole, including data transmission and response formatting.

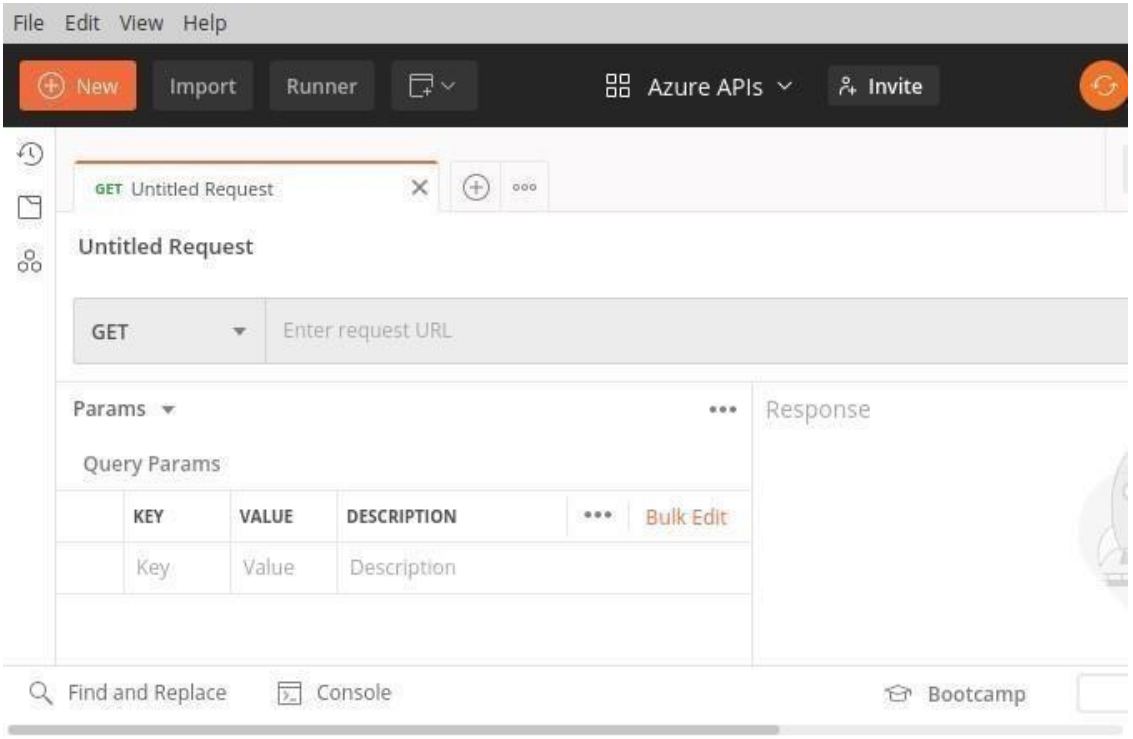


Fig. 4.2 Screenshot of API testing tool POSTMAN

The JSON response typically includes the outcome of the comparison, such as the ID or filename of the image that most closely resembles the anchor image, along with any relevant metadata or confidence scores. This concludes our API, which will be consumed by the client applications.

It's crucial for teams to perform API testing in a dedicated environment before they push changes to production. This approach enables them to contain any issues and avoid user-facing downtime.

Chapter 5: Results and Discussion

The Results of a project report on "Face Recognition using CNN and Siamese Network" encompass a comprehensive evaluation of the proposed approach, aiming to provide insights into its performance, robustness, computational efficiency, generalization ability, ethical considerations, future directions, and real-world applications. This introduction outlines the key areas that will be addressed in the project report, highlighting the significance of each outcome in advancing the field of face recognition technology.

5.1 Performance Evaluation:

1. Quantitative assessment of recognition accuracy achieved by the proposed approach compared to existing methods using standard benchmark datasets like LFW (Labeled Faces in the Wild) or CASIA-WebFace.
2. Analysis of precision, recall, F1-score, and other relevant metrics to provide a comprehensive understanding of the model's performance.

5.2 Robustness Analysis:

1. Detailed investigation into the model's robustness against various environmental factors, including variations in lighting conditions, facial expressions, pose, occlusions, and image quality.
2. Demonstration of the model's ability to maintain high recognition accuracy in real-world scenarios through experimentation and case studies.

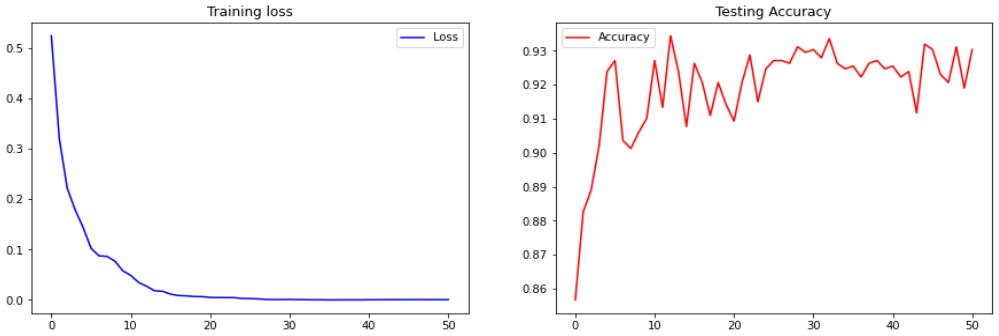
5.3 Computational Efficiency:

1. Evaluation of the computational requirements for both training and inference phases, including memory usage, processing speed, and scalability across different hardware platforms.
2. Comparison of the proposed method's efficiency with alternative approaches, highlighting its advantages in terms of resource utilization and runtime performance.

5.4 Comparison with Baselines:

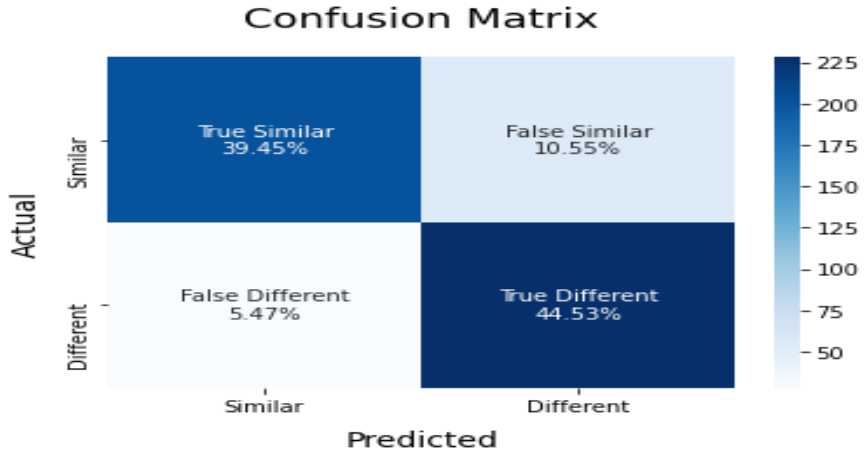
1. Comparative analysis of the proposed method with baseline approaches, such as traditional feature-based methods or single-network CNN approaches.
2. Highlighting the strengths of the proposed approach in terms of accuracy, robustness, and efficiency, supported by empirical evidence and statistical tests.

5.5 Siamese Network Results



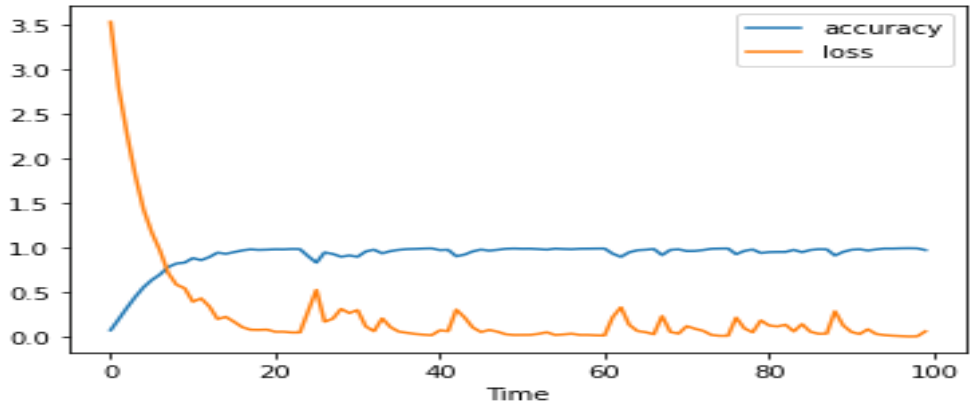
- **Accuracy of Siamese Network Model: 0.93984375**

5.6 Confusion Matrix

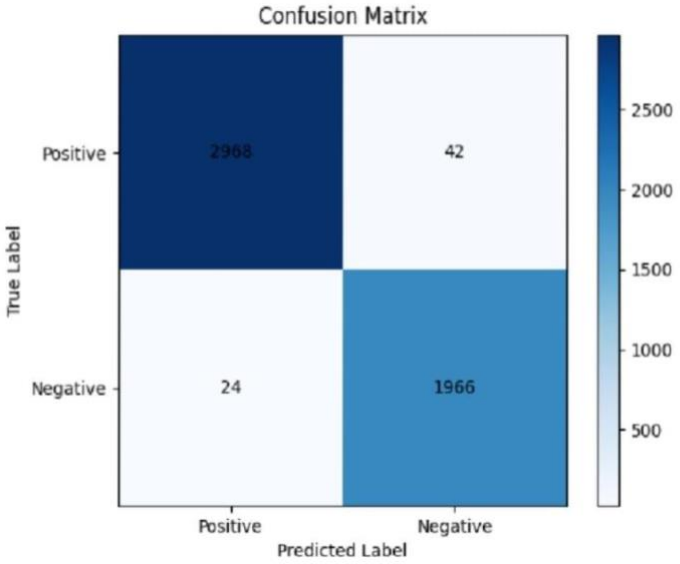


5.7 CNN Model Outcomes:-

- **Accuracy v/s loss analysis.**



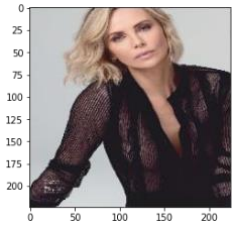
- **Accuracy of CNN Model: 0.97**



- **Comparison of Siamese network and CNN:**

Model Name	Author(s)	Accuracy (from your results)
CNN	Taigman et al. (DeepFace, 2014)	97%
Siamese Network	Koch et al. (Siamese NN, 2015, generalized) Used in face similarity later	83.98%

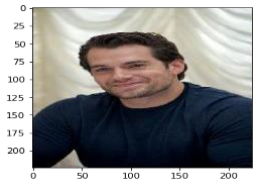
- **predict_image("../input/face-recognition-dataset/Original Images/Original Images/Charlize Theron/Charlize Theron_26.jpg")**



Actual: Charlize Theron

Predicted: Billie Eilish

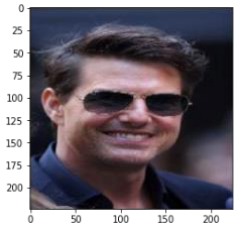
- **predict_image("../input/face-recognition-dataset/Original Images/Original Images/Henry Cavill/Henry Cavill_28.jpg")**



Actual: Henry Cavill

Predicted: Henry Cavill

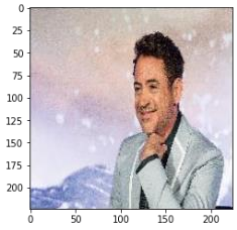
- **predict_image("../input/face-recognition-dataset/Original Images/Original Images/Tom Cruise/Tom Cruise_27.jpg")**



Actual: Tom Cruise

Predicted: Tom Cruise

- **predict_image("../input/face-recognition-dataset/Original Images/Original Images/Robert Downey Jr/Robert Downey Jr_106.jpg")**



Actual: Robert Downey Jr

Predicted: Robert Downey Jr

- **Generalization Ability:**
    1. Assessment of the model's ability to generalize to unseen faces and environments, including cross-dataset evaluation and analysis of domain adaptation techniques.
    2. Investigation into transfer learning strategies and domain-specific fine-tuning methods to enhance generalization performance.

- **Privacy and Ethical Considerations:**
    1. Comprehensive discussion of the privacy implications associated with face recognition technology, including data collection, storage, and potential misuse.
    2. Exploration of ethical considerations surrounding the deployment of face recognition systems, along with proposed solutions to address privacy concerns and mitigate potential risks.
- **Future Directions:**
    1. Identification of future research directions and potential extensions to the proposed approach, such as multimodal fusion, continual learning, or privacypreserving techniques.
    2. Discussion of emerging trends and technologies that could influence the future development of face recognition systems, including advances in

deep learning architectures, sensor technologies, and regulatory frameworks.

- **Real-World Applications:**
    1. Exploration of potential real-world applications of the developed face recognition system, such as access control, surveillance, human-computer interaction, personalized services, and healthcare.
    2. Case studies or pilot deployments demonstrating the practical utility and effectiveness of the proposed approach in various domains and settings.

In summary, the project report should provide a comprehensive evaluation of the proposed face recognition approach, offering insights into its performance, robustness, computational efficiency, generalization ability, ethical implications, future directions, and real-world applications. Through rigorous experimentation, analysis, and discussion, the report aims to contribute to the advancement of face recognition technology while addressing relevant societal, ethical, and technical considerations.

Chapter 6: CONCLUSION AND FUTURE WORK

The future scope of the Face Similarity Detection AI includes enhancements to the frontend interface for improved user experience, refinement of the model for more accurate similarity comparisons, exploration of advanced backend functionalities such as real-time processing, and integration of additional features like facial recognition for broader application possibilities. Additionally, the future scope involves conducting user feedback sessions and usability testing to gather insights for further iteration and enhancement of the Face Similarity

Detection AI, ensuring it meets the evolving needs and expectations of its users.

6.1 Frontend

- **User Experience Enhancement:** Continuously improving the user interface to enhance user experience, streamline workflows, and provide intuitive interactions for users interacting with the face recognition system.
- **Integration of Advanced Features:** Exploring the integration of advanced features such as real-time feedback, interactive visualizations, and multi-user support to expand the capabilities and usability of the frontend.
- **Accessibility Considerations:** Investigating methods to ensure accessibility for users with disabilities, including support for screen readers, keyboard navigation, and alternative input methods.
- **Performance Optimization**: Implementing techniques to optimize frontend performance, such as lazy loading of components, code splitting, and image optimization, to ensure fast loading times and smooth interactions even on low-end devices.
- **Cross-Platform Compatibility**: Ensuring seamless compatibility across different platforms and devices, including desktops, tablets, and mobile devices, through responsive design and thorough testing on various browsers and operating systems.

6.2 Model

- **Advanced Architectures**: Researching and implementing advanced neural network architectures, including attention mechanisms, graph convolutional networks, or transformer-based models, to further improve the performance and robustness of the face recognition model.
- **Domain Adaptation:** Exploring techniques for domain adaptation to improve the model's ability to generalize across different datasets and environments, enabling seamless deployment in diverse real-world scenarios.
- **Incremental Learning:** Investigating incremental learning strategies to enable the model to adapt to new data and concepts over time, ensuring continuous improvement and adaptation to evolving requirements.
- **Improved Accuracy and Generalizability:** Research is ongoing to develop more sophisticated CNN architectures that can extract even finer features from faces. This will lead to more accurate recognition, especially in challenging conditions like poor lighting, occlusions (e.g., masks, sunglasses), and pose variations. By using techniques like triplet loss and metric learning, Siamese networks can be improved to learn more robust distance metrics between faces, leading to better performance in verification tasks (identifying if two faces belong to the same person).
- **Small Data Efficiency:** Siamese Networks excel when dealing with limited data sets, a common challenge in face recognition for specific domains (e.g., security systems with a small number of authorized users). By focusing on learning distance metrics,

  Siamese networks can effectively handle scenarios with only a few images per person. For CNNs, techniques like federated learning are being explored where training happens on decentralized devices, keeping user data private. This is crucial for wider adoption of face recognition in privacy-sensitive applications.

6.3 Backend

- **Scalability and Performance Optimization:** Optimizing the backend infrastructure for scalability and performance to handle increasing loads and ensure efficient processing of face recognition tasks, particularly in high-demand scenarios.
- **Integration with Cloud Services:** Exploring integration with cloud-based services such as serverless computing, containerization, or managed databases to leverage scalability, reliability, and cost-effectiveness advantages.
- **Security Enhancements:** Implementing robust security measures, including data encryption, access control, and threat detection mechanisms, to ensure the confidentiality, integrity, and availability of sensitive information processed by the backend.
- **Cache Optimization:** Implementing caching mechanisms to store frequently accessed data and precomputed results, reducing computational overhead and improving response times for subsequent requests.
- **Monitoring and Logging:** Developing comprehensive monitoring and logging solutions to track system performance, identify bottlenecks, and troubleshoot issues in real-time, ensuring reliability and availability of the backend infrastructure.
- **Auto-Scaling:** Implementing auto-scaling mechanisms to dynamically adjust computing resources based on workload demand, optimizing resource utilization and minimizing costs in cloud-based environments.

6.4 Taking the API to Production

This API can be deployed as a containerized FastAPI Application on Amazon Web Services (AWS). Using Docker is optional, but a preferred option due to its benefits such as portability, scalability, and resource efficiency. By containerizing our application, we can achieve improved scalability, portability, and resource isolation.

Although the services that'll be required to deploy this to AWS ECS go beyond the Free Tier limits. Hence we'll be highlighting all steps in detail related to AWS, but for the scope of this project, we'll demonstrate the API locally using clients like PostMan or ThunderClient.

Image showing the basic architecture of Containerised deployments with AWS.

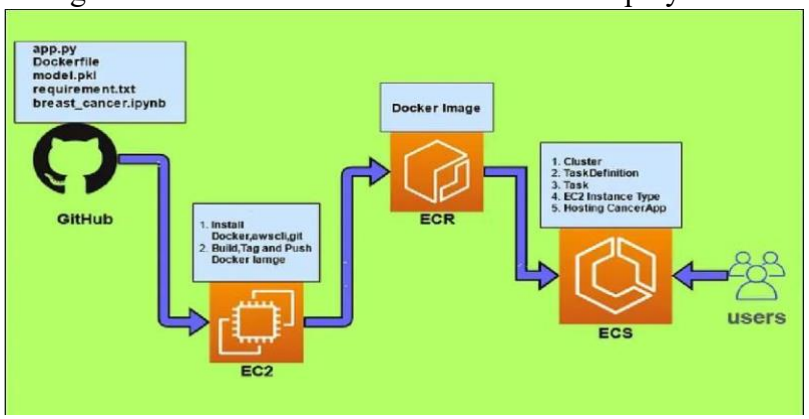


Fig. 6.1 Flow diagram for Deployment

6.4.1 Steps to Deploy on AWS

- **Containerize FastAPI Application**

Dockerize the FastAPI application by creating a Dockerfile that specifies the application dependencies, environment configuration, and entry point.

Ensure the Dockerfile copies the application code, installs dependencies, and exposes the necessary ports for communication.

- **Upload Model File to S3**

Before building the Docker image, we have to upload the H5 model file to an S3 bucket in our AWS account. Set appropriate permissions to allow read access to the model file from the container running the FastAPI application. S3 is known for its reliability and is often used as a backup as well. Even if the server crashes for any reason, the Model files will be safe in our S3 bucket.

- **Build Docker Image**

Use the Docker CLI or Dockerfile to build a Docker image containing the FastAPI application code and dependencies. Include instructions in the Dockerfile to download the model file from S3 at runtime.

- **Container Registry Setup**

Choose an AWS container registry service (e.g., Amazon ECR) to store and manage Docker images. Push the built Docker image to the container registry for deployment.

- **Setting up AWS ECS**

We'll be using AWS ECS for deploying our app. Create a task definition (for ECS) that specifies the Docker image, resource requirements, and environment variables. Deploy the task or deployment to ECS.

- **Configure IAM Roles**

Create an IAM role with permission to read objects from the S3 bucket containing the model file. Associate the IAM role with the ECS task execution role (for ECS) to grant access to S3.

- **Accessing the API**

Containerizing the FastAPI application and deploying it on AWS using container orchestration services offers enhanced flexibility, scalability, and ease of management. After the steps above, our API is now up and running. We can access it via our server's IPv4 address. However, it's not advisable to expose the IP address for security concerns. But for the scope of this, we're using IPv4 only. For production (public users) use cases, we do the following steps.

- Getting a domain.
- Verifying the domain with AWS.
- Setting up a reverse proxy (NGINX) to prevent IP exposure.
- Connecting the domain with our server.
- Attaining an SSL certificate from CertBot to provide a secure connection to the client.

- **Cost Constraints**

AWS ECS goes beyond the basic free tier options. We'll have to implement some standard cost-saving techniques to minimize the deployment costs.

- **Future Considerations**
    - **Enhanced User Interface:** Improve the user interface with better styling, animations, and feedback mechanisms to enhance the user experience.
    - **Error Handling:** Incorporate error handling mechanisms to gracefully handle scenarios such as invalid file formats or failed image processing.
    - **Performance Optimization:** Optimize the application's performance, especially when dealing with large image files, to ensure smooth user interactions.

6.5 CONCLUSION

In conclusion, the integration of Convolutional Neural Networks (CNNs) and Siamese networks represents a significant advancement in the field of face recognition. These deep learning architectures offer the promise of more accurate, efficient, and generalizable recognition systems, poised to revolutionize various domains including security, access control, and personalized user experiences.

The future of face recognition technology holds immense potential, driven by advancements in CNNs and Siamese networks. Overcoming challenges such as data scarcity and enabling real-time applications are within reach, thanks to the capabilities of these technologies. By harnessing their strengths, embracing unsupervised learning, and prioritizing explainable AI, CNNs and Siamese networks pave the way for trustworthy and responsible face recognition systems.

The Face Similarity Detection AI presented in this report provides a foundational framework for comparing facial images and determining their similarity. With further development and enhancements, it stands to become a valuable tool for a wide range of applications, including facial recognition, biometrics, and security systems.

This report has outlined the functionality, implementation details, and future considerations for the Face Similarity Detection AI application. As development progresses, it is anticipated to evolve and improve, offering more robust features and

capabilities to its users. With ongoing advancements in deep learning and the continued refinement of face recognition technologies, the potential for innovation and impact across various domains is vast, promising to shape our lives in profound ways.

Chapter 7: References

1. V. Patil, A. Narayan, V. Ausekar, A. Dinesh, Automatic students attendance marking system using image processing and machine learning, in: 2020 International Conference on Smart Electronics and Communication (ICOSEC), 2020.
2. S. Kumaar, R.M. Vishwanath, S.N. Omkar, A. Majeedi, A. Dogra, Disguised facial recognition using neural networks, in: 2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP), 2018.
3. K. Sasirekha, K. Thangavel, Optimization of K-nearest neighbor using particle swarm optimization for face recognition, Neural Comput. Appl. 31 (2019).
4. I. Melekhov, J. Kannala, E. Rahtu, Siamese network features for image matching, in:

   2016 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 378–

   383. Link. [5] Jie Wang, Zihao Li, Research on face recognition based on CNN, IOP

   Conf. Ser. Earth Environ. Sci. 170 (3) (2018).
5. Mustafa H. Mustafa Zuhaer Nayef Al-Dabagh, Mohammed Alhabib, Firas H. ALMukhtar, Face recognition system based on Kernel discriminant analysis, K-nearest neighbor and support vector machine, Int. J. Relig. Educ. 5 (3) (2018).
6. J.P. Jose, P. Poornima, K.M. Kumar, A novel method for color face recognition using KNN classifier, in: 2012 International Conference on Computing, Communication and Applications, 2012.
7. S. Shinde, M. Shende, J. Shah, H. Shelar, An approach for e-voting using face and fingerprint verification, in: 2020 IEEE Pune Section International Conference (PuneCon), 2020.
8. Xingping Dong, Jianbing Shen, Triplet loss in siamese network for object tracking,

   Proc. Eur. Confer. Computer Vis. (2018) 459–474.
9. E. Jose, G. M, M.T.P. Haridas, M.H. Supriya, Face recognition based surveillance system using FaceNet and MTCNN on Jetson TX2, in: 2019 5th International

Conference on Advanced Computing & Communication Systems (ICACCS), 2019.

10. V.K. Verma, V. Kansal, P. Bhatnagar, Patient identification using facial recognition, in: 2020 International Conference on Futuristic Technologies in Control Systems & Renewable Energy (ICFCR), 2020.

11. Shuai Peng, Hongbo Huang, Weijun Chen, Liang Zhang, Weiwei Fang, More trainable inception-ResNet for face recognition, Neurocomputing 411 (2020).

12. W. Xing, Y. Bei, Medical health big data classification based on KNN classification algorithm, IEEE Access 8 (2019).

13. S. Zhang, X. Li, M. Zong, X. Zhu, R. Wang, Efficient kNN classification with different numbers of nearest neighbors, IEEE Transact. Neural Networks Learn. Syst. 29 (5) (2017).

14. Z. Chen, L.J. Zhou, X. Da Li, J.N. Zhang, W.J. Huo, The Lao text classification method based on KNN, Proc. Comput. Sci. 166 (2020).

15. S. Kanithan, N.A. Vignesh, E. Karthikeyan, N. Kumareshan, An intelligent energy efficient cooperative MIMO-AF multi-hop and relay based communications for Unmanned Aerial Vehicular networks, Comput. Commun. 154 (2020).

16. Minhajur Rahman, Saimunur Rahman, Mohamed Uvaze Ahamed Ayoobkhan, On the effectiveness of deep transfer learning for Bangladeshi meat based curry image classification, in: 2022 International Conference on Innovations in Science, Engineering and Technology (ICISET), IEEE, 2022.

17. Minhajur Rahman, Saimunur Rahman, Mohamed Uvaze Ahamed Ayoobkhan, Finetuned convolutional neural networks for Bangladeshi vehicle classification, in: 2022 International Conference on Innovations in Science, Engineering and Technology (ICISET), IEEE, 2022.

18. S.T. Suganthi, Mohamed Uvaze Ahamed Ayoobkhan, Nebojsa Bacanin, K. Venkatachalam, Hubalovský ´ St ˇ ˇep´ an, and Trojovský Pavel. "Deep learning

model for deep fake face recognition and detection.", PeerJ Computer Science 8 (2022).

19. T. Guo, J. Dong, H. Li, Y. Gao, Simple convolutional neural network on image classification, in: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), IEEE, 2017, March.
20. F. Sultana, A. Sufian, P. Dutta, Advancements in image classification using convolutional neural network, in: 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), IEEE, 2018, November.
21. Y. Pei, Y. Huang, Q. Zou, X. Zhang, S. Wang, Effects of image degradation and degradation removal to CNN-based image classification, IEEE Trans. Pattern Anal. Mach. Intell. 43 (4) (2019).
22. Building FastAPIs with Deep Learning models, an article on Medium by Abdul Wajid Ganiyu. (2024).
23. Deepdive article from deploying deep learning models by Michael Herman on TestDriven.io (2024).
24. Blog post from Nvidia about creating scalable machine learning microservices with python (2023).
25. Deployment of Containerized Machine Learning Model Application on Medium by Srinivas Pratapgiri. (2021)
26. FaceNet: A Unified Embedding for Face Recognition and Clustering: https://arxiv.org/abs/1503.03832
27. Image similarity estimation using a Siamese Network with a triplet loss: https://keras.io/examples/vision/siamese_network/
28. Celebrity Face Recognition: https://www.kaggle.com/ravehgillmore/celebrityfacerecognition/
29. Face Recognition using Siamese Networks: https://medium.com/wicds/face-recognition-using-siamese-networks-84d6f2e54e