**[IIT-Delhi-Artificial Intelligence and Machine Learning For Industry- Batch02](https://lmsportal.timespro.com/course/view.php?id=438)**

Shailesh Ganpat Belose

[shaileshbelose@gmail.com](mailto:shaileshbelose@gmail.com)

+91 9595833595

Contents

[Overview: 1](#_Toc175660504)

[Introduction: 2](#_Toc175660505)

[Dataset Overview 2](#_Toc175660506)

[System Design 2](#_Toc175660507)

[Dependencies 3](#_Toc175660508)

[Model Architecture (Model.py) 4](#_Toc175660509)

[Model Summary 4](#_Toc175660510)

[Netron Visualization of Model 6](#_Toc175660511)

[REST API (restapi.py) 8](#_Toc175660512)

[Testing 8](#_Toc175660513)

[Future Work 8](#_Toc175660514)

[Use Case 9](#_Toc175660515)

[Conclusion 9](#_Toc175660516)

# Overview:

This is a document which will refer Face Recognition System using AI/ML which can help to recognize and classify different images patterns based on labels. For testing data set we are using Labelled Faces in the Wild data set which is public benchmark for use. It has almost 13,000/- images for evaluation.

# Introduction:

This project focuses on building a face recognition system using the Labeled Faces in the Wild (LFW) dataset. The system is designed to identify individuals based on their facial features by leveraging a Convolutional Neural Network (CNN) for feature extraction and classification. The project consists of two main components:

1. **Model Training(Model.py):** This script is responsible for training the CNN model on the LFW dataset.
2. **REST API(restapi.py):** This script provides an API endpoint that allows users to send images for prediction and receive the identified person's name along with a matching score.
3. **Face Image Processor(FaceCropping.ipynb):** In this script we are cropping all images from LFW dataset, Here simply extracting faces from actual images. Using OpenCV with HaarCascades haarcascade\_frontalcatface.xml for achieving this mechanism.

# Dataset Overview

The LFW dataset contains more than 13,000 labeled images of faces from a variety of different people. The images are collected from the internet and come with pre-labeled names. In this project, we use the cropped version of the LFW dataset (LFW Cropped) for training and testing the model.

* **LFW Cropped:** Contains processed images where the faces have been detected and cropped.
* **LFW RAW:** The original, unprocessed images.

# System Design

The system is designed to take advantage of preprocessed images for training and recognition. The overall design can be broken down into the following components:

A diagram of a software flow

Description automatically generated

1. **Image Processor:** This component is responsible for preprocessing the raw images. It uses a Haar Cascade classifier (haarcascade\_frontalface\_default.xml) to detect faces in images and crop them accordingly.
2. **Model Training:** The model is built using TensorFlow and Keras. It is trained to recognize different individuals using the cropped images from the LFW dataset.
3. **REST API:** A Flask-based API is created to handle incoming image requests, preprocess them, and return predictions using the trained model.
4. **Postman Client:** Used for testing the API. An image is sent via POST request, and the API returns the predicted class name along with a matching score.

## Dependencies

* TensorFlow for model creation.
* Flask for building the REST API.
* Postman for testing the API.
* Numpy, CV2 for numerical operations and image processing.
* OS, Json, sklearn

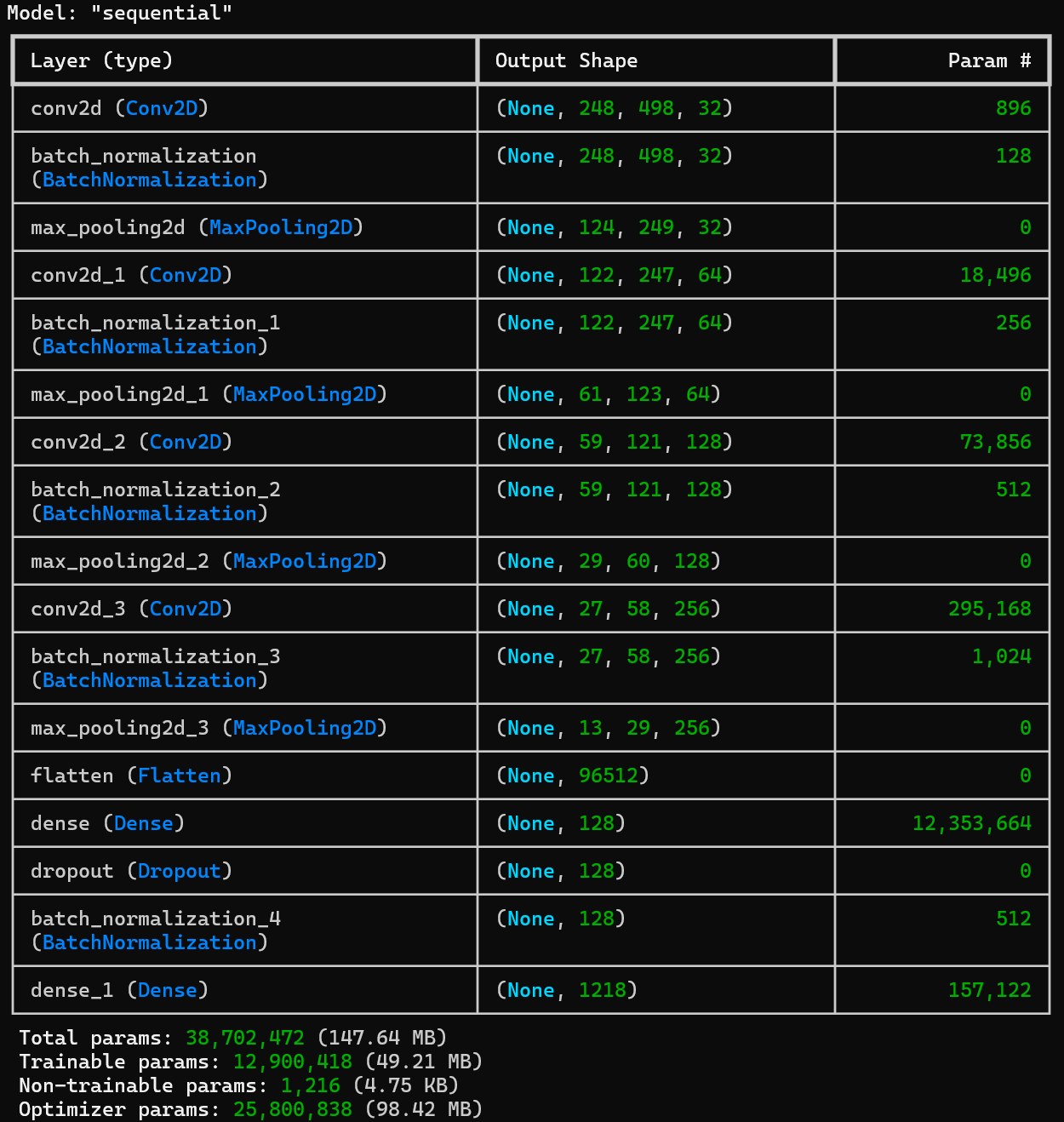
## Model Architecture (Model.py)

The model is a Convolutional Neural Network (CNN) designed to classify faces. Below is a step-by-step breakdown of the model architecture:

* **Input Layer:** The model accepts an input of shape (250, 500, 3) which represents a concatenated image of two 250x250 images (side-by-side) with three color channels (RGB).
* **Convolutional Layers:** The model has four convolutional layers with increasing filter sizes (32, 64, 128, and 256). Each convolutional layer is followed by a Batch Normalization layer and MaxPooling layer for downsampling.
* **Flatten Layer:** The output of the last convolutional layer is flattened to a 1D vector.
* **Fully Connected Layer:** A dense layer with 128 neurons is used for further processing, followed by a dropout layer to prevent overfitting.
* **Output Layer:** The final dense layer has neurons equal to the number of unique classes in the dataset, with a softmax activation function to output the probability distribution over the classes.
* **Loss Function:** Sparse categorical cross-entropy is used since the labels are integers.
* **Optimizer:** The Adam optimizer is used for training.

The model is trained over 40 epochs with a batch size of 32, and the best-performing model is saved as FRSv\_croppedv1.1.keras.

### Model Summary



### Netron Visualization of Model

## REST API (restapi.py)

The REST API is implemented using Flask. The main functionality includes:

* **Loading the Model:** The trained model (FRSv\_croppedv1.1.keras) is loaded once when the Flask server starts.
* **Class Names:** The class names are loaded from the class\_names.json file, which maps class indices to the names of the individuals.
* **Image Preprocessing:** Incoming images are resized to match the model's expected input size (500x250 pixels) and normalized.
* **Prediction:** The processed image is fed into the model, and the predicted class index is obtained. This index is mapped to the corresponding name from the class\_names.json file.
* **Response:** The API returns the predicted class name along with a matching score in JSON format.

# Testing

Testing is done using Postman, where images are sent to the API and the predicted class name and matching score are returned. The model's predictions are checked against the known names to ensure accuracy.



# Future Work

**Model Improvement:** Explore more complex model architectures or fine-tuning pre-trained models like VGG16 or ResNet for better performance.

**Scalability:** Implement the API using a more robust framework or deploy it in a cloud environment for better scalability.

**Additional Features:** Add functionalities such as image augmentation, real-time face recognition, and improved preprocessing techniques.

# Use Case

* Public place surveillance system like schools, colleges, government offices
* Forensic department police use case for riots, rally's, festivals etc

# Conclusion

This project successfully demonstrates the creation of a face recognition system using deep learning techniques. The system is able to classify images from a well-known dataset with a high degree of accuracy, and the REST API provides an easy-to-use interface for making predictions.