MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE

(Autonomous)

NEW PANVEL

PROJECT REPORT ON

"Face Mask Detection Using CNN"

IN PARTIAL FULFILLMENT OF

MASTER OF DATA ANALYTICS (PART II)

SEMESTER IV – 2024-25

PROJECT GUIDE

Prof. Omkar Sherkhane

SUBMITTED BY: ASHWIN SURESH

ROLL NO: 6855

Mahatma Education Society's

PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE

(Autonomous)

Re-accredited "A" Grade by NAAC (3rd Cycle)

Project Completion Certificate

THIS IS TO CERTIFY THAT

Ashwin Suresh

of **M.Sc. Data Analytics Part - II** has completed the project titled **"Face Mask Detection Using CNN"** of subject **Deep Learning** under our guidance and supervision during the academic year 2024-25 in the department of Data Analytics.

Project Guide	Course Coordinator	Head of the Department

Pillai College of Arts, Commerce & Science (Autonomous) New Panvel

CA – 2 Project

Face Mask Detection Using CNN

Introduction

The COVID-19 pandemic has underscored the importance of public health safety measures, among which wearing face masks has played a critical role in minimizing the spread of the virus. As societies strive to return to normalcy, enforcing mask mandates in public areas has become a necessity. However, manual monitoring of mask compliance is both inefficient and prone to error. To address this challenge, this project presents an automated **Face Mask Detection System** using deep learning and computer vision techniques.

This system is designed to detect whether individuals are wearing a face mask or not in real-time using a webcam feed. Built using Python and TensorFlow/Keras, the system leverages a Convolutional Neural Network (CNN) model trained on a labeled dataset containing images of people with and without face masks. The model is capable of learning key visual patterns and features that distinguish between masked and unmasked faces.

The real-time detection module employs OpenCV for face detection, using Haar cascades to locate facial regions in video frames. Once a face is detected, it is preprocessed and fed into the trained model, which predicts the presence or absence of a face mask. The system then visually annotates the video feed, marking faces with green rectangles for "MASK" and red for "NO MASK," accompanied by appropriate labels and timestamp overlays.

This project serves as an example of how artificial intelligence and machine learning can contribute to public health and safety. It can be deployed in various environments such as hospitals, schools, transportation hubs, and office buildings to ensure compliance with safety protocols without human intervention.

Key Features

1. **Model Persistence and Training**
    - **Load or Train:** The script first checks if a pre-trained model file exists. If available, it loads the model and any saved training history; otherwise, it builds and trains a new CNN model from scratch.
    - **Checkpointing:** It uses a model checkpoint callback that saves the best model based on validation accuracy, ensuring that training improvements are captured and stored.
2. **CNN-Based Mask Classification**
    - **Architecture:** The CNN is built using multiple convolutional layers with ReLU activations and max pooling, followed by dense layers. The final layer uses a sigmoid activation function for binary classification (mask vs. no mask).
    - **Loss and Optimizer:** The model is compiled with binary cross-entropy as the loss and the Adam optimizer to efficiently learn from the training data.
3. **Data Augmentation**
    - **ImageDataGenerator:** The training data is augmented on-the-fly via operations like rescaling, shearing, zooming, and horizontal flipping. This

process enriches the dataset, enabling the model to generalize better on unseen images.
    - **Separate Pipelines:** A dedicated data generator is also set up for the test/validation set, applying only rescaling to maintain data integrity.
4. **Real-Time Face Detection and Prediction**
    - **Live Video Capture:** The project leverages OpenCV to capture a live video feed through a webcam using the DirectShow backend.
    - **Face Detection with Haar Cascades:** It employs a pre-trained Haar Cascade classifier to detect faces within each video frame.
    - **Face Processing:** For every detected face, a cropped image is extracted and preprocessed (resized, converted to an array) for the model's prediction.
    - **Mask Determination:** A threshold (0.5 by default) is used to decide whether a face is wearing a mask or not.
5. **User Interface and Visual Feedback**
    - **Annotations:** The system draws real-time bounding boxes around detected faces. It uses green for "MASK" and red for "NO MASK" to provide immediate visual feedback.
    - **Timestamp Overlay:** The current date and time are overlaid on the video feed, adding context to the detection process.

Code Walkthrough

Facemask.py

- **Dependencies & Setup:** The script imports necessary libraries like OpenCV for image processing and TensorFlow/Keras for building the CNN model.

- **Model Management:** It checks if a pre-trained model and its training history exist. If found, the model is loaded; otherwise, a CNN is built and trained from scratch with data augmentation.

- **Data Handling:** Training and test images are loaded via Keras' ImageDataGenerator, which also applies transformations (e.g., rescaling, shearing) to boost generalization.

- **Training with Checkpoints:** The model is trained for a fixed number of epochs with ModelCheckpoint saving the best version based on validation accuracy.

- **Real-Time Detection:** Using OpenCV's Haar Cascade, the system captures live video, detects faces, preprocesses them, and uses the trained model to predict mask usage.

- **Visual Feedback:** Detected faces are annotated with colored rectangles (green for "MASK", red for "NO MASK"), and a timestamp is overlaid on the frame.

- **Resource Cleanup:** The script ends by releasing the webcam and closing all OpenCV windows.

Code

facemask.py

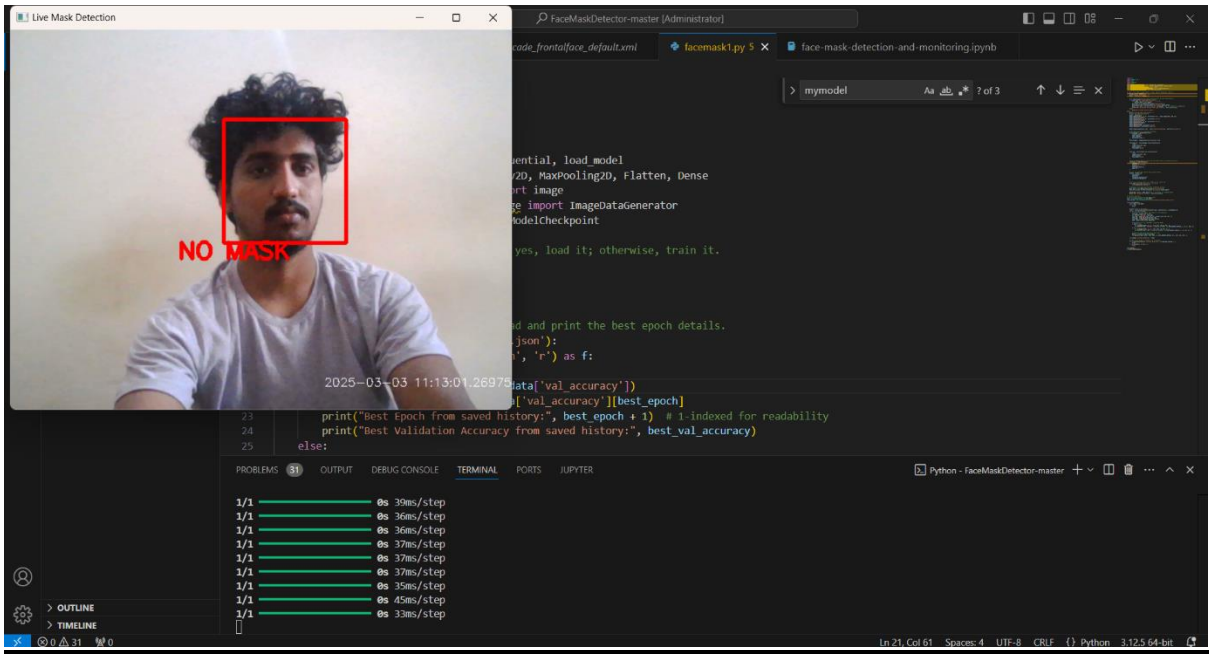
```
import os
import numpy as np
import cv2
import datetime
import json
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint

# Check if the model already exists. If yes, load it; otherwise, train it.
if os.path.exists('mymodel.h5'):
    print("Loading saved model...")
    model = load_model('mymodel.h5')

    # If training history was saved, load and print the best epoch details.
    if os.path.exists('training_history.json'):
        with open('training_history.json', 'r') as f:
            history_data = json.load(f)
        best_epoch = np.argmax(history_data['val_accuracy'])
        best_val_accuracy = history_data['val_accuracy'][best_epoch]
        print("Best Epoch from saved history:", best_epoch + 1)  # 1-indexed for readability
        print("Best Validation Accuracy from saved history:", best_val_accuracy)
    else:
        print("No training history found.")
else:
    print("Training model from scratch...")
    # Build the model architecture
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
    model.add(MaxPooling2D())
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D())
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

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

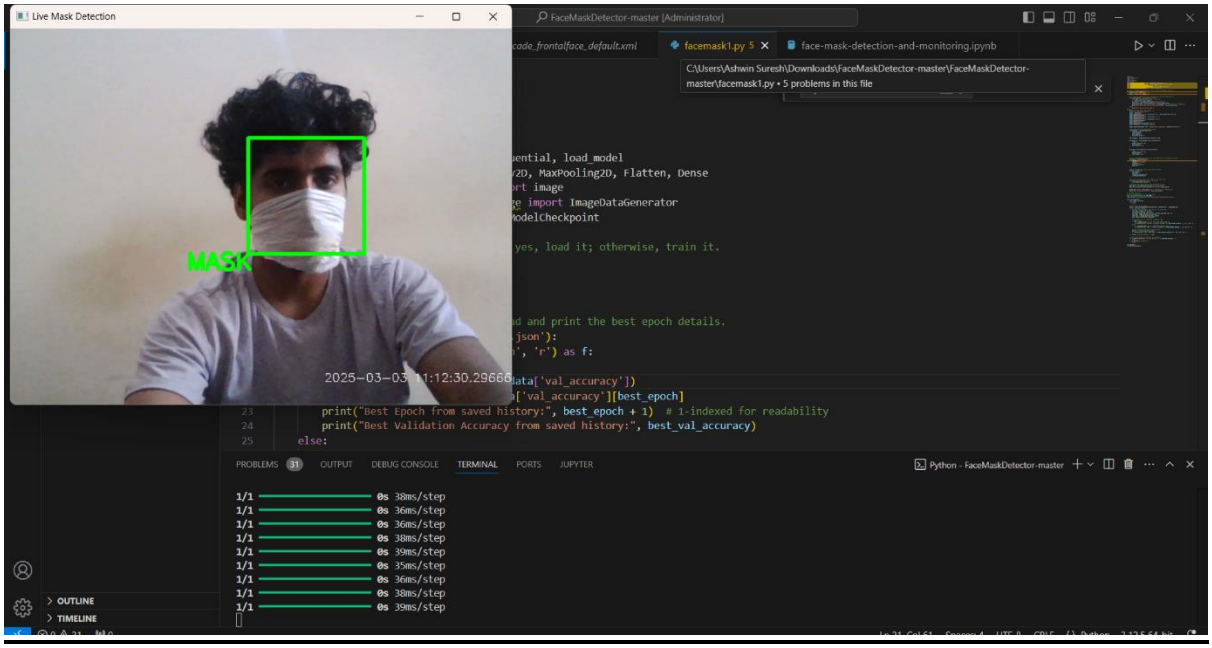
```
# Data generators for training and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory(
    'train',
    target_size=(150, 150),
    batch_size=16,
    class_mode='binary'
)

test_set = test_datagen.flow_from_directory(
    'test',
    target_size=(150, 150),
    batch_size=16,
    class_mode='binary'
)

# Set up ModelCheckpoint to save the best model based on validation accuracy
checkpoint = ModelCheckpoint(
    'mymodel.h5',
    monitor='val_accuracy',
    verbose=1,
    save_best_only=True,
    mode='max'
)

# Train the model and capture the training history
history = model.fit(
    training_set,
    epochs=10,
    validation_data=test_set,
    callbacks=[checkpoint]
)

# Save the training history to a JSON file for future use
with open('training_history.json', 'w') as f:
    json.dump(history.history, f)

# Determine which epoch had the best validation accuracy
best_epoch = np.argmax(history.history['val_accuracy'])
```

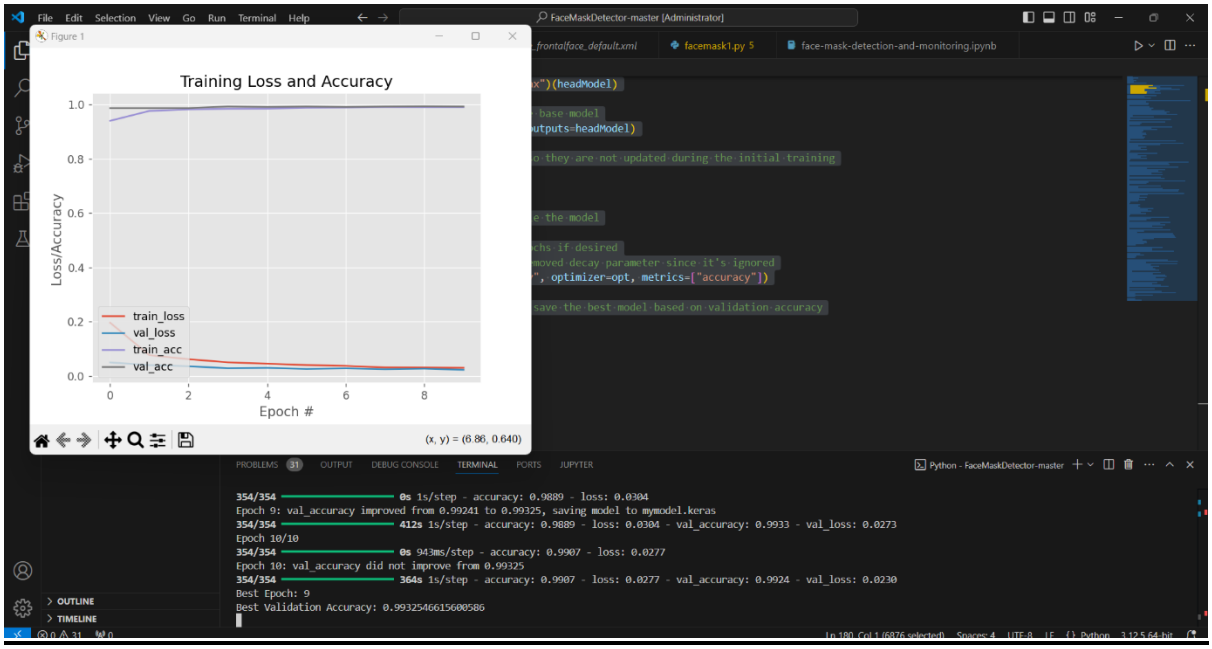
```
    best_val_accuracy = history.history['val_accuracy'][best_epoch]

    print("Best Epoch:", best_epoch + 1)  # Converting to 1-indexed count
    print("Best Validation Accuracy:", best_val_accuracy)

# Live Mask Detection
# Use DirectShow backend to avoid MSMF issues
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(frame, scaleFactor=1.1, minNeighbors=4)
    for (x, y, w, h) in faces:
        # Extract the face region and prepare it for prediction
        face_img = frame[y:y+h, x:x+w]
        cv2.imwrite('temp.jpg', face_img)
        test_img = image.load_img('temp.jpg', target_size=(150, 150, 3))
        test_img = image.img_to_array(test_img)
        test_img = np.expand_dims(test_img, axis=0)
        pred = model.predict(test_img)[0][0]

        # Threshold: >= 0.5 is "NO MASK", otherwise "MASK"
        if pred >= 0.5:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 3)
            cv2.putText(frame, 'NO MASK', ((x+w)//2, y+h+20),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 3)
        else:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 3)
            cv2.putText(frame, 'MASK', ((x+w)//2, y+h+20), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 3)

        # Display current date and time on the frame
        datet = str(datetime.datetime.now())
        cv2.putText(frame, datet, (400, 450), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,
255), 1)

    cv2.imshow('Live Mask Detection', frame)

    # Break if the window is closed or 'q' is pressed
    if cv2.getWindowProperty('Live Mask Detection', cv2.WND_PROP_VISIBLE) < 1:
        break
```

```
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Output:







Conclusion

This Face Mask Detection project successfully demonstrates the power and potential of integrating deep learning with real-time computer vision. By leveraging a convolutional neural network for classification and OpenCV for live face detection, the system effectively distinguishes between masked and unmasked faces, providing immediate visual feedback. The project not only addresses a critical public health need by automating mask compliance monitoring but also showcases how modern machine learning techniques can be utilized to build practical, real-world applications. The modular design—from loading or training the model, applying data augmentation, to processing live video streams—ensures both scalability and adaptability, paving the way for future enhancements and deployment in various public settings. Ultimately, this project serves as a valuable prototype that can be further optimized and integrated into larger public safety and surveillance systems.