FACE MASK DETECTION SYSTEM

-SAKSHI

CONTENTS

OVERVIEW OF FACE MASK DETECTION SYSTEM

WHAT:

This project is a computer vision application that detects whether a person is wearing a mask or not and stores data of those not wearing masks.

Computer Vision & Machine Learning

- This project falls under computer vision, a domain of machine learning that enables computers to process, analyse, and interpret visual data from images and videos.
- It uses deep learning algorithms trained on large datasets to improve accuracy in detecting masks in different environments and lighting conditions.

Data Sources & Collection

- The application gathers data from multiple sources, including:
  - IP cameras for real-time surveillance.
  - Webcams for direct monitoring through computers.
  - Online video feeds for remote or cloud-based analysis.
- The system can process both live feeds and pre-recorded footage to detect mask usage.

Web-Based Application & Accessibility

- This is a web-based application, making it easily accessible over a LAN (Local Area Network) or an internet connection for remote access.
- It can be integrated with cloud storage or local databases to store and analyse data over time.

Integration with IP Cameras & Network Setup

- IP cameras are wireless surveillance cameras that transmit footage over a Wi-Fi network.
- Many IP cameras have built-in hotspot functionality, allowing them to function as independent networks.
- If the computer running the application is connected to the same Wi-Fi network as the camera, it can process real-time footage to detect mask violations.
- The application can be configured to work with multiple cameras simultaneously for large-scale monitoring.

WHY:

Health & Safety Compliance

- Helps enforce mask-wearing in critical environments like hospitals, research labs, and industrial areas.
- Reduces the risk of exposure to harmful chemicals, pollutants, and airborne pathogens.
- Aids in preventing the spread of infectious diseases such as COVID-19.

Automation & Efficiency

- Provides an automated solution for real-time face mask detection, reducing the need for manual monitoring.
- Enhances workplace and public safety by ensuring compliance with health regulations.

Technological Skill Development

- Demonstrates proficiency in machine learning**,** computer vision**,** and deep learning techniques.
- Showcases expertise in Python**,** OpenCV**,** Keras**,** and web development**.**
- Strengthens problem-solving and critical thinking skills.

Real-World Impact

- Can be integrated with CCTV cameras, access control systems, and IoT devices for smart surveillance.

How:

Backend

- Establish a connection with an IP camera to capture real-time video.
- Perform face detection using OpenCV to identify individuals.
- Use Keras and NumPy for mask detection, classifying people as wearing or not wearing masks.
- Save detected face data, including images of individuals, for further analysis.

Frontend

- Develop a web application using Streamlit for real-time mask detection display.
- Design a user-friendly interface to show detected faces and their mask status.

PREREQUISITES

Here's a list of prerequisites for building a Face Mask Detection System. These include both software and hardware requirements as well as knowledge areas:

➢ KNOWLEDGE PREREQUISITES

- Python Programming – Basic to intermediate level
- Computer Vision – Understanding image processing concepts
- Machine Learning / Deep Learning – Especially CNNs (Convolutional Neural Networks)
- Data Pre-processing – Image resizing, normalization, augmentation
- Model Evaluation – Accuracy, precision, recall, F1-score, etc.

➢ SOFTWARE REQUIREMENTS

1. Programming Language:

    - Visit the official Python website: Download Python

2. Libraries/Frameworks:

    - OpenCV – For image and video processing
    - TensorFlow / Keras – For building and training the model
    - NumPy – For numerical operations
    - SciPy – For metrics and utility functions
    - Pillow – For image loading, saving, and basic manipulations
    - Tempfile – For creating and handling temporary files securely.

➢ DATASET REQUIREMENTS

A dataset with labelled images of:

- People wearing face masks
- People not wearing face masks

➢ HARDWARE REQUIREMENTS

- Webcam / Camera – For real-time face mask detection

CREATE VIRTUAL ENVIRONMENT

A virtual environment (venv) is an isolated workspace for Python projects. It allows you to install dependencies separately from the system-wide Python installation, preventing conflicts between different projects.

Features of Virtual Environment:

- Dependency Management – Keeps project-specific libraries separate.
- Avoid Conflicts – Different projects can use different versions of the same package.
- Reproducibility – Makes it easier to share projects with others.
- Cleaner Development – Prevents clutter in the global Python environment.

Steps for Creating a Virtual Environment (V.E)

- Go to C Drive → Create a Project Folder
  Inside the Project Folder, create a new folder named Facemask.
- This Facemask folder will act as the V.E (Virtual Environment).
- You can create multiple virtual environments for different projects in this Project Folder.

Now, Create the Virtual Environment

- Copy the address: C:\Projects\Facemask
- Open Command Prompt (cmd)
- Write the address in cmd or navigate to the folder manually
- Run the following command:

```
python -m venv C:\Projects\Facemask
```

- If successful, you will see the virtual environment created inside the Facemask folder

Inside the Facemask Folder, you should see:

- Include
- lib
- Scripts

Activate Virtual Environment:

- Navigate to the Scripts folder inside your Facemask folder, Click on the address bar and type cmd.
- Run the following command: activate

```
C:\Projects\Facemask\Scripts>activate
```

Important Notes:

- You must activate the virtual environment every time you want to use it
- If you don't activate it, Python will use the global environment instead of the virtual one
- If activated correctly, all installed packages will be saved in the virtual environment, not in the main system

Installation of OpenCV Package

The OpenCV (Open-Source Computer Vision Library) package is widely used for face detection due to its powerful image processing capabilities. To install OpenCV, open a terminal or command prompt and run:

```
(Facemask) C:\Projects\Facemask\Scripts>pip install opencv-python
```

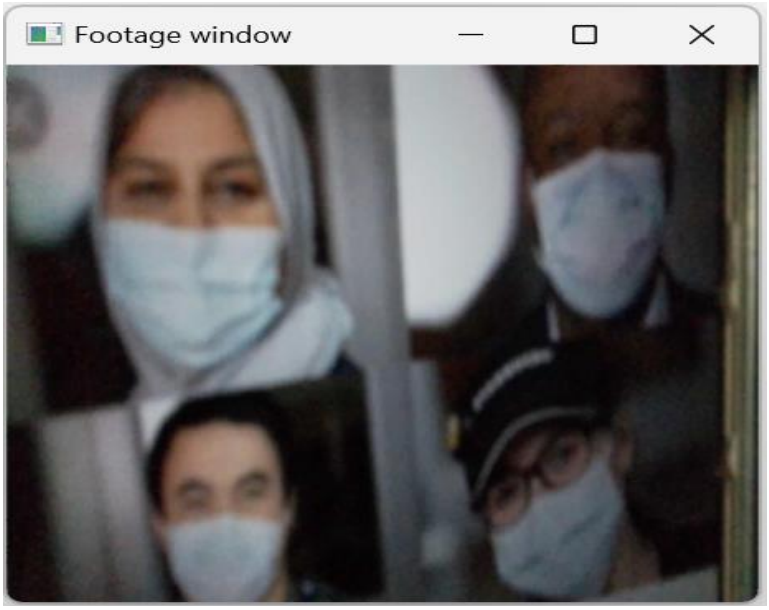
REAL TIME VIDEO STREAMING FROM AN IP CAMERA USING OPENCV

Open the Backend.py file in your in-code editor, and add the following code for real-time video streaming.

```
import cv2
# Capture video stream from the provided IP camera URL
vid=cv2.VideoCapture("http://192.110.49.10:8080/video")
while (vid.isOpened()):   # Read a frame from the video stream
    flag,frame=vid.read()
    if(flag):
        # Create a resizable window for displaying the footage
        cv2.namedWindow("Footage window" ,cv2.WINDOW_NORMAL)
         # Display the captured frame in the window
        cv2.imshow("Footage window",frame)
        key=cv2.waitKey(20)
        if(key==ord('x')): # If the 'x' key is pressed, exit the loop
            break
    else:   # If the frame is not captured, exit the loop
        break
vid.release() # Release the video capture object
cv2.destroyAllwindows() # Close all OpenCV windows
```

- This code captures a live video stream from an IP camera using OpenCV.
- It continuously reads frames and displays them in a resizable window, named Footage Window.
- The user can exit the video stream by pressing the 'x' key. After exiting, the video capture is released, and all OpenCV windows are closed.

Here's a video footage window



Final Output:

- A resizable window named "Footage window" opens, displaying a live video.
- The video updates continuously frame by frame as long as the camera is streaming.
- If the 'x' key is pressed, the video stream stops, and the window closes.
- If the script fails to capture a frame (e.g., due to a connection issue or incorrect URL), the loop exits automatically.
- Once the script exits, the video capture is released, and all OpenCV windows are closed

FACE DETECTION USING PYTHON

Face detection is a key technology in computer vision, widely used in security systems, social media applications, and human-computer interactions. Python provides multiple libraries to perform face detection efficiently.

➢ Adding a Rectangle to Video Before Face Detection

Before implementing face detection, it's essential to understand how to process video frames and draw basic shapes like rectangles. This helps in visualizing object detection areas before applying complex algorithms.

When working with video streams, each frame is processed individually. By drawing a rectangle, we can mark specific regions of interest, which is useful in face detection and other computer vision tasks.

Why Add a Rectangle?

- Understanding Video Processing: Helps in handling video frames effectively.
- Marking Areas of Interest: Useful for pre-processing before detection.
- Testing Drawing Functions: Ensures that detected regions are correctly visualized.

Once we successfully overlay rectangles on video frames, we can move forward with face detection by integrating libraries.

Open the Backend.py file in your in-code editor, and add the following code for adding a rectangle in a video:

```
import cv2
# Capture video stream from the provided IP camera URL
vid=cv2.VideoCapture("Video.mp4")
i=1
while (vid.isOpened()):   # Read a frame from the video stream
    flag,frame=vid.read()
    if(flag):
        # Draws a black rectangle on the frame.
        cv2.rectangle(frame,(800+i,650),(1200+i,1050),(0,0,0),6)
        i=i+5
    # Create a resizable window for displaying the footage
        cv2.namedWindow("Footage window" ,cv2.WINDOW_NORMAL)
    # Display the captured frame in the window
        cv2.imshow("Footage window",frame)
        key=cv2.waitKey(20)
        if(key==ord('x')): # If the 'x' key is pressed, exit the loop
            break
    else:   # If the frame is not captured, exit the loop
        break
vid.release  # Release the video capture object
cv2.destroyAllWindows() # Close all OpenCV windows
```

- Loads a video file and processes it frame by frame.
- Draws a moving black rectangle on each frame, shifting it horizontally.
- Displays the updated frame in a resizable window.
- Waits for user input, allowing the video to stop when 'x' is pressed.
- Releases resources and closes the window after the video ends.

Here's a video footage window with rectangle in it.

Final Output:

- A video window opens, showing the video footage.
- A black rectangle moves horizontally across the screen.
- The user can press 'x' to stop the video.

➢ FACE DETECTION

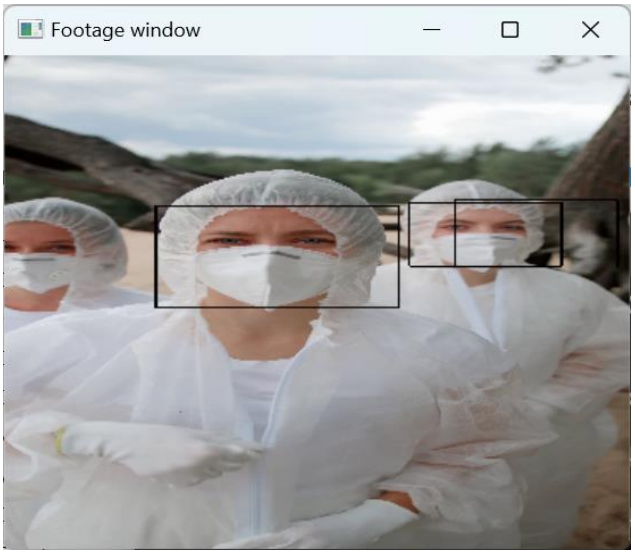
Now, let's take the next step—detecting faces in real time! We'll use OpenCV's built-in face detection features to identify and highlight faces in each frame. This will allow us to track faces dynamically as the video plays.

Open the Backend.py file in your in-code editor, and add the following code for detecting Faces in a video:

```
import cv2
#Defining the model
facemodel=cv2.CascadeClassifier("face.xml")
# Capture video stream
vid=cv2.VideoCapture("Video.mp4")
i=1
while (vid.isOpened()):    # Read a frame from the video stream
    flag,frame=vid.read()
    if(flag):
        faces=facemodel.detectMultiScale(frame)
        for (x,y,l,w) in faces:
            # Draws a black rectangle on the frame.
            cv2.rectangle(frame,(x,y),(x+l,y+w),(0,0,0),6)
        # Create a resizable window for displaying the footage
        cv2.namedWindow("Footage window" ,cv2.WINDOW_NORMAL)
    # Display the captured frame in the window
        cv2.imshow("Footage window",frame)
        key=cv2.waitKey(20)
        if(key==ord('x')): # If the 'x' key is pressed, exit the loop
            break
    else:    # If the frame is not captured, exit the loop
        break
vid.release  # Release the video capture object
cv2.destroyAllWindows() # Close all OpenCV windows
```

- Loads OpenCV for image and video processing.
- Uses a pre-trained face detection model (face.xml).
- Opens and reads a video file (Video.mp4).
- Processes each frame to detect faces.
- Draws black rectangles around detected faces.
- Displays the processed video in a resizable window.
- Allows the user to exit by pressing 'x'.
- Stops if a frame cannot be read.
- Releases resources and closes all windows after execution.

Here's a video footage window with rectangle in it.



Final Output:

- The system detected multiple faces in the image, highlighting them with black rectangles.
- However, one face on the left side was not detected.
- This suggests the detection is mostly effective, with minor room for improvement.

➢ DETECT THE FACES AND SAVE THEIR DATA

Inside the scripts directory of the virtual environment, create a dedicated folder named "data" to store and organize the images efficiently. This folder will serve as a designated location for saving all image files, ensuring better management and accessibility within the virtual environment's structure.

Open the Backend.py file in your in-code editor, and add the following code for saving the data of detected Faces:

```
import cv2
#Defining the model
facemodel=cv2.CascadeClassifier("face.xml")
# Capture video stream
vid=cv2.VideoCapture("http://192.189.143.13:8080/video")
i=1
while (vid.isOpened()):   # Read a frame from the video stream
    flag,frame=vid.read()
    if(flag):
        faces=facemodel.detectMultiScale(frame)
        for (x,y,l,w) in faces:
            # Extract the face region from the frame
            face_img=frame[y:y+w,x:x+l]
              # Define the file path to save the detected face image
            path="data/"+str(i)+".jpg"
            i=i+1
            cv2.imwrite(path,face_img)
        # Draws a black rectangle on the frame.
            cv2.rectangle(frame,(x,y),(x+l,y+w),(0,0,0),6)
        # Create a resizable window for displaying the footage
        cv2.namedWindow("Footage window" ,cv2.WINDOW_NORMAL)
    # Display the captured frame in the window
        cv2.imshow("Footage window",frame)
        key=cv2.waitKey(20)
        if(key==ord('x')): # If the 'x' key is pressed, exit the loop
            break
    else:   # If the frame is not captured, exit the loop
        break
vid.release  # Release the video capture object
cv2.destroyAllWindows() # Close all OpenCV windows
```

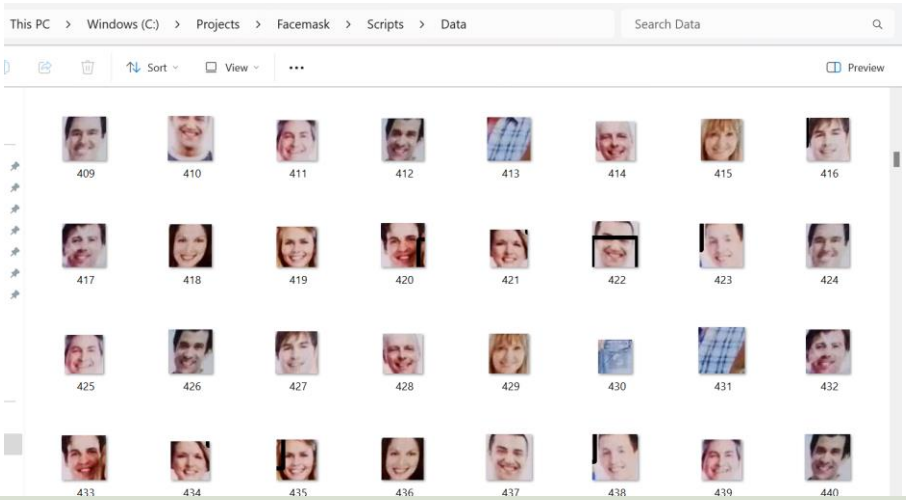
- The script captures a live video stream from an external source, such as an IP camera or a mobile phone acting as a webcam.
- A pre-trained face detection model is loaded to identify faces within the frames.
- The model scans the image, recognizing facial features based on patterns learned during training.
- Whenever a face is detected, it is extracted from the frame and stored as an image file in a specific directory.
- A black rectangle is drawn around each detected face to provide a visual indication of face detection.
- The script listens for user input, allowing the user to stop the video stream by pressing the 'x' key.
- All OpenCV windows are closed at the end of execution to ensure a proper shutdown of the application.

Here's a live video footage window with detected faces.

Final Output:

- The output is a real-time video feed with detected faces highlighted by black rectangles.
- Detected faces are extracted and saved as image files in the data/ directory with sequential names.
- The video updates dynamically, showing face detections as people move in front of the camera.
- Pressing 'x' stops the program, closes windows, and releases system resources.

Detected faces are extracted and saved as image files in the data folder



FACE MASK DETECTION MODEL: TRAINING PHASE

The training phase is a crucial step in building a Face Mask Detection Model as it allows the model to learn patterns and features that differentiate between masked and unmasked faces. This phase involves feeding the model with labelled image data, enabling it to recognize and classify images correctly.

Role of Key Libraries in Training:

- **TensorFlow** – A deep learning framework used to build and train the model with layers like convolutional neural networks (CNNs) for feature extraction.
- **Pillow (PIL)** – A Python imaging library that helps in preprocessing images, such as resizing, converting formats, and enhancing image quality.
- **SciPy** – Used for advanced image manipulations like filtering, transformations, and augmentation to improve model performance.

Install these Libraries by writing this following code:

open a terminal or command prompt and run:

```
(Facemask1) C:\Projects\Facemask1\Scripts>pip install pillow
(Facemask1) C:\Projects\Facemask1\Scripts>pip install scipy
(Facemask1) C:\Projects\Facemask1\Scripts>pip install tensorflow
```

To train a Face Mask Detection model, you need a dataset containing images of people wearing masks and without masks. These images serve as training data, allowing the model to learn the differences between the two categories. The dataset is typically organized into two folders:

- With Mask – Contains images of people wearing masks.
- Without Mask – Contains images of people without masks.

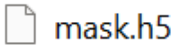
By storing training data and scripts in the virtual environment, we ensure a structured and reproducible training process. This setup allows easy deployment of the model for real-time face mask detection

Open the train.py file in your in-code editor, and add the following code for training the model:

```
# Importing necessary layers
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense
from keras.models import Sequential  #Importing Sequential model from Keras
from tensorflow.keras.optimizers import Adam # Importing Adam optimizer
# Importing ImageDataGenerator for data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Define the Model
mymodel=Sequential()  # Initializing the sequential model
# First convolutional layer with 32 filters
mymodel.add(Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
mymodel.add(MaxPooling2D())# First max pooling layer to reduce spatial dimensions
mymodel.add(Conv2D(32,(3,3),activation='relu'))
mymodel.add(MaxPooling2D())
mymodel.add(Conv2D(32,(3,3),activation='relu'))
mymodel.add(MaxPooling2D())
mymodel.add(Flatten())# Flattening the output to feed into the dense layer
mymodel.add(Dense(100,activation="relu")) # Fully connected layer with 100 neurons
mymodel.add(Dense(1,activation="sigmoid"))# Output layer with sigmoid activation for binary classification
mymodel.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy']) # Compile the Model

#Define the Data
# Data augmentation for training data
train=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test=ImageDataGenerator(rescale=1./255) # Rescaling test data without augmentation
# Loading training images
train_img=train.flow_from_directory('train',target_size=(150,150),batch_size=16,class_mode="binary")
# Loading testing images
test_img=test.flow_from_directory('test',target_size=(150,150),batch_size=16,class_mode="binary")
#Train and Test the model
mask_model=mymodel.fit(train_img,epochs=10,validation_data=test_img)
#Save the model in my directory
mymodel.save('mask.h5',mask_model)
```

- Imports Required Libraries – Keras and TensorFlow modules for building and training a CNN, along with data augmentation tools.
- Defines a CNN Model – Uses multiple Conv2D layers with ReLU activation, MaxPooling, Flatten, and Dense layers for feature extraction and classification. The final layer uses sigmoid activation for binary classification.
- Compiles the Model – Uses the Adam optimizer and binary cross-entropy loss to optimize performance, tracking accuracy as a metric.
- Prepares Data – Applies data augmentation (rescaling, shear, zoom, flipping) to training images and rescaling only for test images.
- Loads Training & Test Data – Reads images from directories, setting a target size of 150x150 and a batch size of 16 for binary classification.
- Trains the Model – Runs for 10 epochs, validating on the test dataset.
- Saves the Model – The trained model is saved as "mask.h5" for future use.

Saved model in scripts folder of virtual environment



Here's a model training output

```
Epoch 1/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 488ms/step - accuracy: 0.7071 - loss: 0.6515C:\Projects\Facemask1\Lib\site-packages\keras\src\trainers\
data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its construc
tor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()
73/73 ━━━━━━━━━━━━━━━━━━━━ 43s 547ms/step - accuracy: 0.7085 - loss: 0.6487 - val_accuracy: 0.9639 - val_loss: 0.1247
Epoch 2/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 20s 267ms/step - accuracy: 0.9586 - loss: 0.1300 - val_accuracy: 0.9845 - val_loss: 0.1048
Epoch 3/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 21s 283ms/step - accuracy: 0.9784 - loss: 0.0825 - val_accuracy: 0.9742 - val_loss: 0.0762
Epoch 4/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 20s 270ms/step - accuracy: 0.9697 - loss: 0.0791 - val_accuracy: 0.9742 - val_loss: 0.0880
Epoch 5/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 19s 266ms/step - accuracy: 0.9750 - loss: 0.0671 - val_accuracy: 0.9845 - val_loss: 0.0881
Epoch 6/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 19s 266ms/step - accuracy: 0.9778 - loss: 0.0741 - val_accuracy: 0.9639 - val_loss: 0.1040
Epoch 7/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 20s 269ms/step - accuracy: 0.9846 - loss: 0.0484 - val_accuracy: 0.9691 - val_loss: 0.1402
Epoch 8/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 19s 266ms/step - accuracy: 0.9551 - loss: 0.1187 - val_accuracy: 0.9845 - val_loss: 0.0805
Epoch 9/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 20s 268ms/step - accuracy: 0.9852 - loss: 0.0338 - val_accuracy: 0.9639 - val_loss: 0.1212
Epoch 10/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 20s 278ms/step - accuracy: 0.9875 - loss: 0.0331 - val_accuracy: 0.9845 - val_loss: 0.0778
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
nsidered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(
model, 'my_model.keras')`.
```

Final Output:

- Training Accuracy Improves- Starts at 70.71% (Epoch 1) and reaches 98.75% (Epoch 10), indicating the model is learning well.
- Training Loss Decreases- Drops from 0.6515 (Epoch 1) to 0.0331 (Epoch 10), showing the model is minimizing classification errors.
- High Validation Accuracy- Stays around 96.39% to 98.45%, suggesting good generalization to unseen data.

FACE MASK DETECTION MODEL: TESTING PHASE

This program tests a face mask detection model using OpenCV and Keras. It first loads pre-trained models for face detection (Haar Cascade) and mask detection (deep learning model). The program then reads a test image, detects faces, and processes each face by resizing and converting it into a format suitable for the mask detection model. The model predicts whether a person is wearing a mask or not, and the result is displayed with a bounding box around detected faces. Finally, the processed image is shown, highlighting the detected faces.

Open the detect.py file in your in-code editor, and add the following code for face mask detection:

```
import cv2 # OpenCV for image processing
from keras.models import load_model # To load the pre-trained model
from keras.utils import load_img, img_to_array   # For image processing
import numpy as np # For numerical operations

# To read an image and show it.
facemodel = cv2.CascadeClassifier("face.xml")
maskmodel = load_model("mask.h5") # Load the pre-trained mask detection model
img = cv2.imread("childmask.JPG") # Read the input image
face = facemodel.detectMultiScale(img) # Detect faces in the image

for (x, y, w, h) in face: # Loop through each detected face
    crop_face = img[y:y + w, x:x + h]  # Crop the detected face from the image
    cv2.imwrite("temp.jpg", crop_face)# Save the cropped face temporarily
      # Load and resize the cropped face to match the model's input size
    crop_face = load_img("temp.jpg", target_size=(150, 150, 3))
    crop_face = img_to_array(crop_face)  # Convert image to array
    crop_face = np.expand_dims(crop_face, axis=0)
    pred = maskmodel.predict(crop_face)  # Make prediction using the mask detection model
    print(pred)  # Print the prediction results
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,0,0),4) # Black rectangle

cv2.namedWindow("mywindow", cv2.WINDOW_NORMAL) # Create a resizable window
cv2.imshow("mywindow", img)
cv2.waitKey(0)  # Wait for a key press and then close the window
cv2.destroyAllWindows()
```

- Import Libraries – Load OpenCV for image processing, Keras for deep learning, and NumPy for numerical operations.
- Load Pre-Trained Models – Load the Haar Cascade classifier (face.xml) for face detection and the deep learning model (mask.h5) for mask detection.
- Read Image & Detect Faces – Load the test image and use the Haar Cascade model to detect faces.
- Process Each Face – Crop the detected faces, resize them to match the model's input size, convert them to arrays, and prepare them for prediction.
- Predict Mask Status – The deep learning model analyzes each face and predicts:
- 0 = Mask detected
- 1 = No mask detected
- Display Results – Draw rectangles around detected faces, label them based on the prediction, and show the processed image in a window.

Here's a window showing detected Face with mask


```
(Facemask1) C:\Projects\Facemask1\Scripts>detect.py
2025-04-04 11:50:25.177123: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numer
ical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable
 `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-04 11:50:26.664639: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numer
ical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable
 `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-04 11:50:31.128977: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available
 CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow wi
th the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until yo
u train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
[[0.]]
```

Final Output:

Model has successfully processed an input image and generated the following result:

- Predicted Class: 0
- Meaning: "Wear a Mask"
- Raw Model Output: [[0.]]

This means that the model has classified the detected face as someone who is wearing a mask.

FACE MASK DETECTION USING COLOURED RECTANGLES

This Python script detects whether a person is wearing a mask using OpenCV for face detection and a deep learning model for classification. The detected faces are processed, classified, and highlighted with color-coded rectangles:

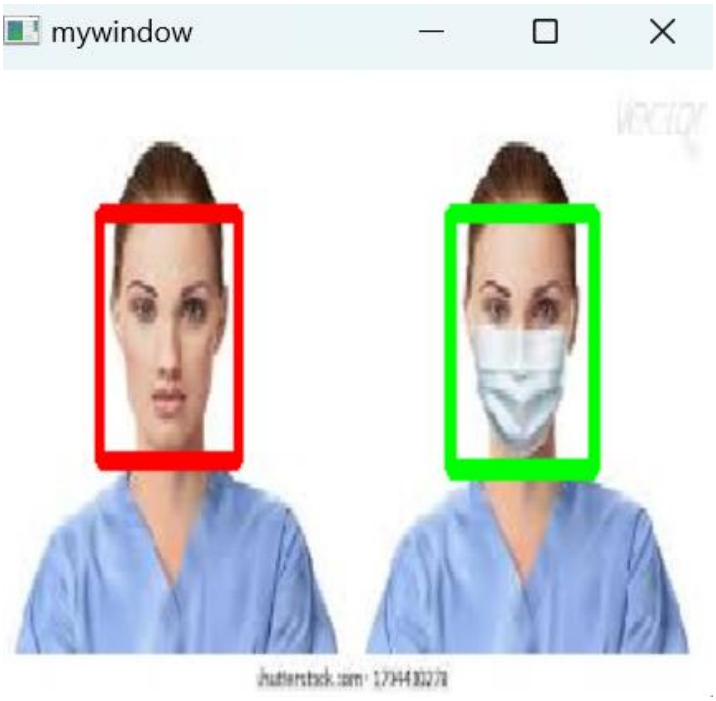
- 🟥 Red Rectangle → No Mask Detected
- 🟩 Green Rectangle → Mask Detected

Open the detect.py file in your in-code editor, and edit the following code for coloured rectangles

```
import cv2  # Import OpenCV for image processing
from keras.models import load_model # Import function to load trained model
from keras.utils import load_img, img_to_array  # Functions for image preprocessing
import numpy as np  # Import NumPy for array manipulation

# To read an image, detect mask and show it.
facemodel = cv2.CascadeClassifier("face.xml")
maskmodel = load_model('mask.h5') # Load the pre-trained mask detection deep learning model

img = cv2.imread("mask.JPG") # Read the input image where mask detection will be performed
face = facemodel.detectMultiScale(img) # Detect faces in the image using the face detection model

for (x, y, l, w) in face: # Loop through each detected face
    crop_face = img[y:y+w, x:x+l]
    cv2.imwrite("temp.jpg", crop_face)

    crop_face = load_img("temp.jpg", target_size=(150, 150, 3))
    crop_face = img_to_array(crop_face)    # Convert the image to a NumPy array for model processing
    crop_face = np.expand_dims(crop_face, axis=0)

    pred = maskmodel.predict(crop_face)[0][0]# Make a prediction using the trained mask detection model

    if pred == 1:
        cv2.rectangle(img, (x, y), (x+l, y+w), (0, 0, 255), 4)  # Red rectangle for no mask
    else:
        cv2.rectangle(img, (x, y), (x+l, y+w), (0, 255, 0), 4)  # Green rectangle for mask

cv2.namedWindow("mywindow", cv2.WINDOW_NORMAL)
cv2.imshow("mywindow", img) # Show the image with the detected faces and rectangles
cv2.waitKey(0)
cv2.destroyAllWindows() # Close all OpenCV windows
```

- Import Libraries – OpenCV, Keras, and NumPy.
- Load Models – Face detection (face.xml) and mask detection (mask.h5).
- Read Image & Detect Faces – Load mask.JPG and identify faces.
- Crop & Preprocess Faces – Resize, convert to an array, and prepare for model input.
- Make Prediction – Detect whether a face is wearing a mask or not.
- Draw Rectangles – Red for no mask, Green for masked faces.
- Display Image – Show results in an OpenCV window.
- Close Windows – Wait for user input and close OpenCV windows.

Here's a window showing coloured rectangles



Final Output of the Face Mask Detection Model:

The system successfully detects whether a person is wearing a mask and highlights the results using color-coded rectangles:

- Left Face 🟥 → No Mask Detected (Red Rectangle)
- Right Face 🟩 → Mask Detected (Green Rectangle)

➢ REAL-TIME FACE MASK DETECTION USING COLOURED RECTANGLES

The script captures video from an IP camera, processes each frame, detects faces, classifies them using a pre-trained model (mask.h5), and displays the processed video with bounding boxes around detected faces. The results are visually indicated using color-coded rectangles:

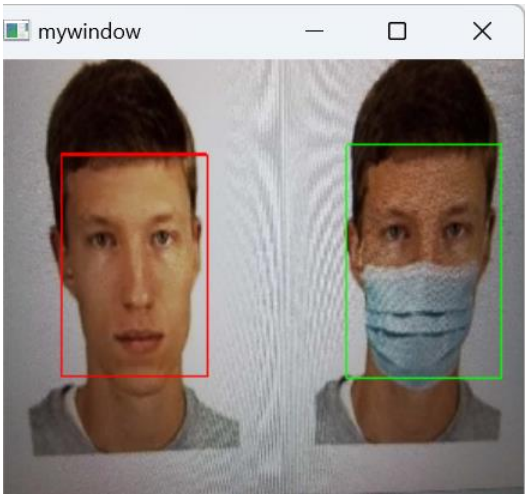
- 🟥 Red Rectangle → No Mask Detected
- 🟩 Green Rectangle → Mask Detected

Open the facetime.py file in your in-code editor, and edit the following code for coloured rectangles using IP camera footage.

```
import cv2# Import OpenCV for image processing
from keras.models import load_model# Import function to load trained model
from keras.utils import load_img, img_to_array# Functions for image preprocessing
import numpy as np# Import NumPy for array manipulation

# To read an image detect mask and show it.
facemodel = cv2.CascadeClassifier("face.xml")
maskmodel = load_model("mask.h5") # Load pre-trained face mask detection model
# Capture video from an IP camera (change the URL to your device's IP stream)
vid = cv2.VideoCapture('http://132.368.49.95:8080/video')

while(vid.isOpened()):# Start video processing loop
    flag, frame = vid.read()
    if flag:
        face = facemodel.detectMultiScale(frame)# Detect faces in the current frame
        for (x, y, l, w) in face:  # Loop through each detected face
            crop_face = frame[y:y+w, x:x+l]# Crop the detected face from the frame
            cv2.imwrite('temp.jpg', crop_face) # Save the cropped face as a temporary image
            crop_face = load_img('temp.jpg', target_size=(150, 150, 3))
            crop_face = img_to_array(crop_face)# Convert image to an array for model processing
            crop_face = np.expand_dims(crop_face, axis=0)
            pred = maskmodel.predict(crop_face)[0][0]
            if pred == 1:
                cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 0, 255), 4)#red rectangle if no mask is detected
            else:
                cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 255, 0), 4)#green rectangle if no mask is detected
        cv2.namedWindow("mywindow", cv2.WINDOW_NORMAL) # Create a resizable window for displaying results
        cv2.imshow("mywindow", frame)
        cv2.waitKey(15)
    else:
        break
cv2.destroyAllWindows() # Close all OpenCV windows when the video processing ends
```

- Import Libraries – Load OpenCV, Keras, and NumPy.
- Load Models – Use Haar Cascade (face.xml) for face detection and a trained deep learning model (mask.h5) for mask classification.
- Capture Video – Read frames from an IP camera or webcam.
- Detect Faces – Identify faces using the Haar Cascade model.
- Preprocess Faces – Crop, resize (150x150), and convert to an array for model input.
- Predict Mask Status –
    - 0 → Mask Detected (Green Rectangle)
    - 1 → No Mask (Red Rectangle)
- Display Video Output – Show processed frames in a window.
- End Program – Stop video processing and close all windows.

Here's a window streaming coloured rectangles



Final Output:

The displayed output shows the real-time face mask detection results using OpenCV and Deep Learning. The system processes an image or video frame and identifies whether a person is wearing a mask or not.

- 🟥 Red Rectangle (Left Face) – The person is not wearing a mask, indicating a potential safety risk.
- 🟩 Green Rectangle (Right Face) – The person is wearing a mask, ensuring compliance with safety guidelines.

➢ REAL-TIME FACE MASK DETECTION & SAVING UNMASKED FACES

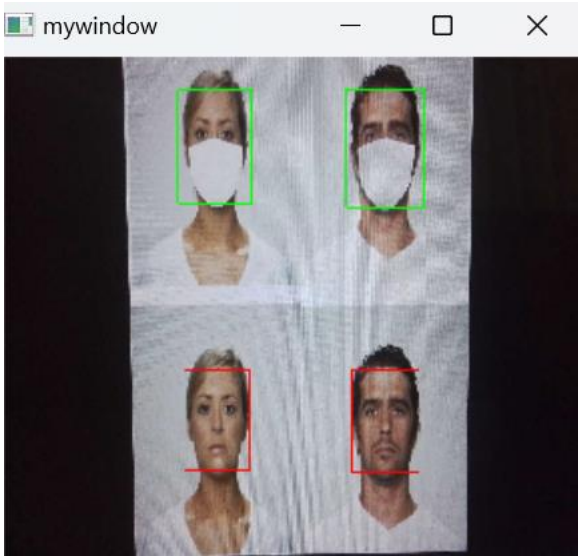
This project implements a real-time face mask detection system using OpenCV and Deep Learning. It detects faces in a live video stream, determines whether a person is wearing a mask, and highlights them with a color-coded bounding rectangle: Additionally, the system automatically saves images of faces not wearing masks in a designated folder (facesdata/).

Open the facetime.py file in your in-code editor, and edit the following code for saving the images of unmasked people:

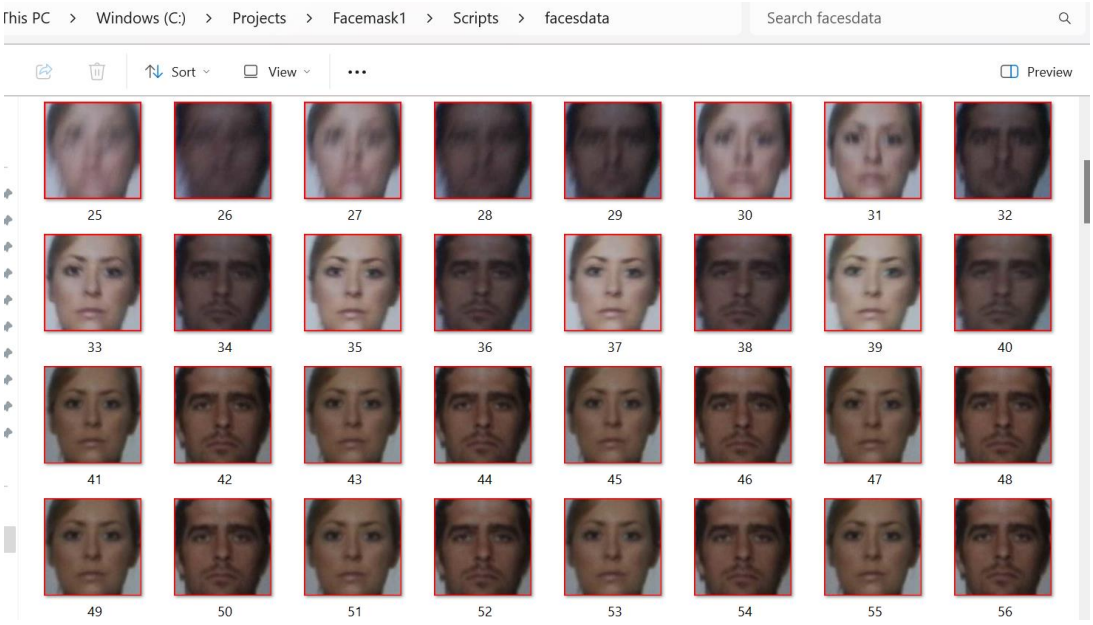
```
import cv2
from keras.models import load_model
from keras.utils import load_img, img_to_array
import numpy as np

# To read an image, detect a mask, and show it.
facemodel = cv2.CascadeClassifier("face.xml")  # Load face detection model
maskmodel = load_model("mask.h5")  # Load trained mask detection model
vid = cv2.VideoCapture("http://192.168.29.13:8080/video")  # Load video file for processing

i = 1  # Counter for saving images

while vid.isOpened():
    flag, frame = vid.read()  # Read a frame from the video
    if flag:
        face = facemodel.detectMultiScale(frame)  # Detect faces in the frame
        for (x, y, l, w) in face:
            crop_face1 = frame[y:y+w, x:x+l]  # Crop detected face
            cv2.imwrite("temp.jpg", crop_face1)  # Save temporary face image
            crop_face = load_img("temp.jpg", target_size=(150, 150, 3))  # Resize image
            crop_face = img_to_array(crop_face)  # Convert image to array
            crop_face = np.expand_dims(crop_face, axis=0)  # Expand dimensions
            pred = maskmodel.predict(crop_face)[0][0]  # Make a prediction

            if pred == 1:
                cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 0, 255), 4)  # Draw red box (no mask)
                path = "facesdata/" + str(i) + ".jpg"  # Save path for detected face
                cv2.imwrite(path, crop_face1)  # Save cropped face image
                i += 1  # Increment image counter
            else:
                cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 255, 0), 4)  # Draw green box (mask)

        cv2.namedWindow("mywindow", cv2.WINDOW_NORMAL)  # Create window
        cv2.imshow("mywindow", frame)  # Display processed frame
        k=cv2.waitKey(15)  # Wait for a short time before next frame
        if (k==ord('x')):
            break
    else:
        break  # Stop processing if video ends

cv2.destroyAllWindows()  # Close all OpenCV windows
```

- Import Libraries – Load OpenCV, Keras, and NumPy.
- Load Models – Face detection (face.xml) and mask detection (mask.h5).
- Capture Video – Connect to an IP camera for live streaming.
- Detect Faces – Identify faces in each video frame.
- Predict Mask Status – Classify faces as masked (green box) or unmasked (red box).
- Save Unmasked Faces – Store images of faces without masks in the "facesdata/" folder.
- Display Output – Show the processed video with bounding boxes.
- Exit on Key Press – Press 'x' to stop the program and close all windows.

Here's a window streaming coloured rectangles



Here's a folder with unmasked face images



Final Output:

- The script detects faces without masks in a live video feed.
- Unmasked faces are saved in the "facesdata" folder.
- Each face is stored as an image with a red rectangle box.

FRONTEND DEVELOPMENT

Frontend development is crucial for a Face Mask Detection System because it provides an interactive and user-friendly interface for users, administrators, and stakeholders. A good frontend enables quick data visualization, real-time mask detection results, and seamless user interactions.

➢ Frontend in Streamlit:

Streamlit is an open-source Python framework for building data-driven web applications. It is simple and efficient, allowing developers to build interactive and real-time web apps quickly using Python. In a face mask detection system, Streamlit can be used to create an interface that shows live results, handles video input, and displays mask detection outputs in a clean and organized manner.

Install Streamlit:

Go to scripts folder of virtual environment and open command prompt and run the following command:

```
(Facemask1) C:\Projects\Facemask1\Scripts>pip install streamlit
```

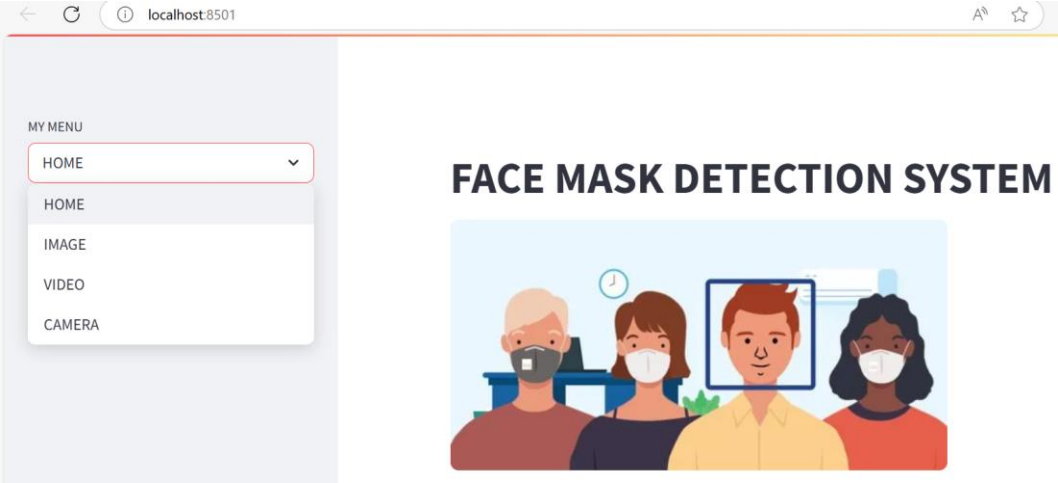
Now after successful installation of streamlit, Open the main.py file in your in-code editor, and add the following basic Streamlit code:

```
import streamlit as st
st.title("Face Mask Detection System") # Set the title for the web application
st.image("https://5.imimg.com/data5/PI/FD/NK/SELLER-5866466/images-500x500.jpg", width=500)
choice=st.sidebar.selectbox("MY MENU",["HOME","IMAGE","VIDEO","Camera"])# Sidebar menu for navigation
```

- Imports the Streamlit library as st.
- Sets the app title to "Face Mask Detection System" using st.title().
- Displays an image from a given URL with a width of 500 pixels using st.image().
- Adds a sidebar with a select box labeled "MY MENU" using st.sidebar.selectbox().
- The sidebar menu allows navigation between HOME, IMAGE, VIDEO, and CAMERA.

Now that you've successfully created your Streamlit app with the title, it opens in the browser and lets you interact with the UI.

Here's a streamlit with title, image and the side menu bar:



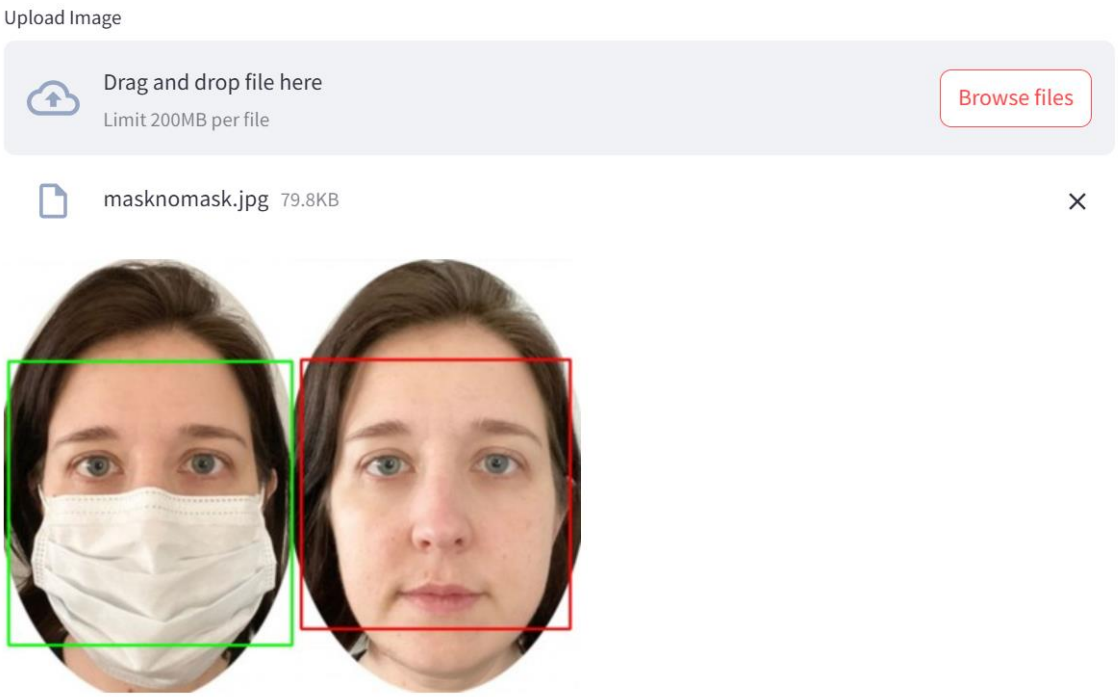
➢ FACE MASK DETECTION FROM UPLOADED IMAGE USING STREAMLIT

Open the main.py file in your in-code editor, and add the following code for face mask detection from uploaded image:

```
# Import necessary libraries
import streamlit as st
from keras.models import load_model#To load the pre-trained mask detection model
from keras.utils import load_img, img_to_array  # For image preprocessing
import numpy as np  # For numerical operations
import cv2 # For image processing and face detection

# Load face detection model and mask detection mode
facemodel=cv2.CascadeClassifier("face.xml")
maskmodel=load_model('mask.h5')

st.title("Face Mask Detection System") # Set the title for the web application
st.image("https://5.imimg.com/data5/PI/FD/NK/SELLER-5866466/images-500x500.jpg", width=500)
choice=st.sidebar.selectbox("MY MENU",["HOME","IMAGE","VIDEO","Camera"])# Sidebar menu for navigation

if(choice=="HOME"):
    st.header("WELCOME")
elif(choice=="IMAGE"):
    file=st.file_uploader("Upload Image")

    if file:
        b=file.getvalue()# Convert the uploaded image into an array
        d=np.frombuffer(b,np.uint8)
        img=cv2.imdecode(d,cv2.IMREAD_COLOR)# Decode the image
        face=facemodel.detectMultiScale(img)# Detect faces in the uploaded image

        for(x,y,w,h) in face:  # Iterate over each detected face
            crop_face=img[y:y+h,x:x+w]
            cv2.imwrite('temp.jpg',crop_face)
             # Load the cropped face image for prediction
            crop_face=load_img('temp.jpg',target_size=(150,150,3))
            crop_face=img_to_array(crop_face)
            crop_face=np.expand_dims(crop_face,axis=0)
            pred=maskmodel.predict(crop_face)[0][0]

            if pred==1:  # Draw a rectangle around the face
                cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),4) # Red rectangle for no mask
            else:
                cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),4)  # Green rectangle for no mask

        st.image(img,channels='BGR',width=400)          # Display the resulting image with rectangles around faces
```

- Import Libraries – Load Streamlit for the app, Keras for the mask model, OpenCV for image processing, and NumPy for array handling.
- Load Models – Load a face detection model (face.xml) and a pre-trained mask detection model (mask.h5).
- Set Up UI – Display the app title, a sample image, and a sidebar menu with navigation options: HOME, IMAGE, VIDEO, and CAMERA.
- HOME Option – Shows a simple welcome message when selected.
- IMAGE Option – Lets the user upload an image, then detects faces in it using OpenCV.
- Face Analysis – Each face is cropped, pre-processed, and passed to the model to check if a mask is worn.
- Draw Results – Red rectangle is shown for faces without a mask, green for faces with a mask.
- Final Output – The processed image with highlighted faces is displayed on the app screen.
- Real-time Feedback – Helps users quickly know who is wearing a mask and who isn't.

Here's an Uploaded image in streamlit to detect the face mask:



Final Output:

- Two faces are detected in the uploaded image.
- Green Rectangle (Left Face): Indicates that the person is wearing a mask.
- Red Rectangle (Right Face): Indicates that the person is not wearing a mask.
- The rectangles are drawn based on the predictions made by the trained mask detection model.

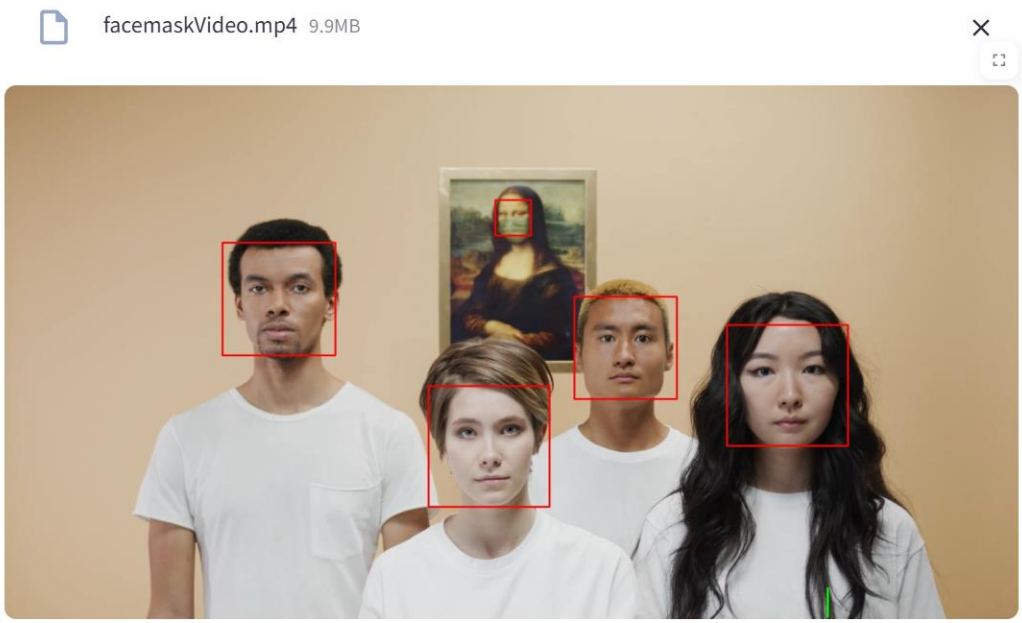
➢ FACE MASK DETECTION FROM UPLOADED VIDEO USING STREAMLIT

Open the main.py file in your in-code editor, and add the following code for face mask detection from uploaded video, it also saves images of the unmasked faces:

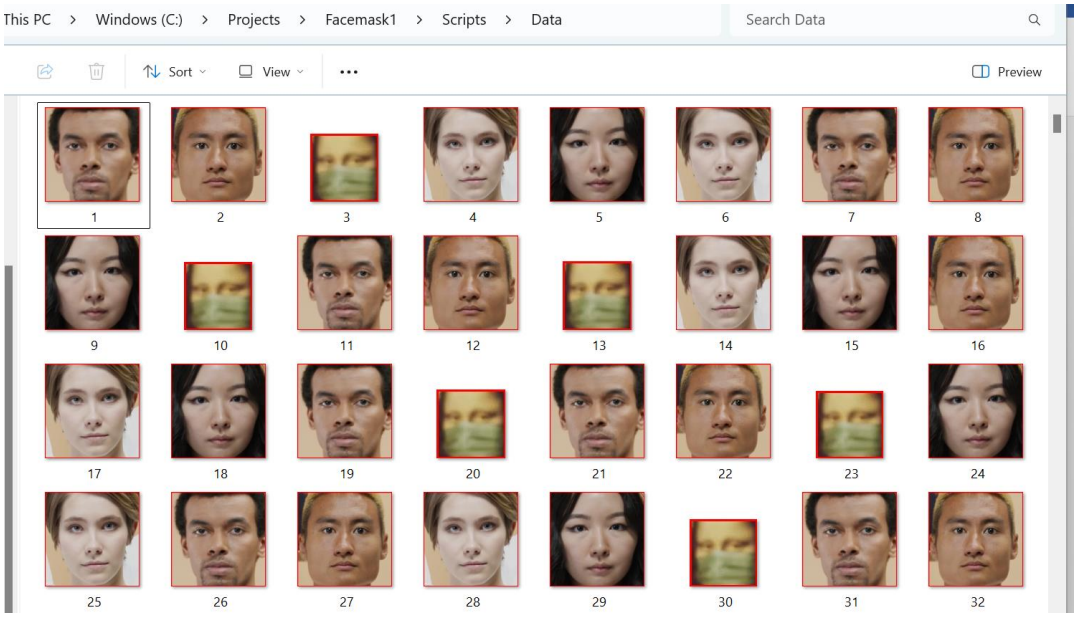
```
elif(choice=="VIDEO"):
    file=st.file_uploader("Upload Video") # Upload a video file
    window = st.empty() # Create an empty space for displaying video frames

    if file:
        tfile=tempfile.NamedTemporaryFile()# Save uploaded video temporarily
        tfile.write(file.read())
        vid=cv2.VideoCapture(tfile.name)# Load video from temporary file
        i=1

        while(vid.isOpened()):  # Read video frame by frame
            flag, frame=vid.read()
            if flag:
                face=facemodel.detectMultiScale(frame)  # Detect faces in the frame

                for (x, y, l, w) in face:
                    crop_face1=frame[y:y+w, x:x+l] #Crop detected face region
                    cv2.imwrite('temp.jpg', crop_face1) # Save temporary cropped face
                    # Preprocess face for prediction
                    crop_face=load_img('temp.jpg', target_size=(150, 150, 3))
                    crop_face=img_to_array(crop_face)
                    crop_face=np.expand_dims(crop_face, axis=0)
                    pred=maskmodel.predict(crop_face)[0][0]

                    if pred==1:  # Predict mask (1 = No Mask, 0 = Mask)
                        cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 0, 255), 4)
                        path = "data/" + str(i) + ".jpg"
                        cv2.imwrite(path, crop_face1) # Save face with no mask
                        i +=1
                    else:
                         #Draw green rectangle for mask
                        cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 255, 0), 4)

                window.image(frame, channels='BGR')# Display the frame with rectangles
```

- The user uploads a video file using the file uploader in the sidebar.
- The uploaded video is saved temporarily using Named Temporary File.
- OpenCV reads the video from the temporary file using cv2.VideoCapture.
- The code enters a loop to read video frames one by one.
- For each frame, faces are detected using the pre-trained face detection model.
- Each detected face region is cropped from the frame and saved temporarily as temp.jpg.
- The cropped face image is preprocessed: resized, converted to array, and reshaped for prediction.
- The mask detection model predicts whether the person is wearing a mask or not.
- If the prediction is 1 (no mask), a red rectangle is drawn and the face image is saved in the data/ folder.
- If the prediction is 0 (mask), a green rectangle is drawn around the face.
- The processed frame, with colored rectangles, is displayed on the app using window.image.

Here's an Uploaded Video in streamlit to detect the face mask:



Here's a folder with unmasked face images:



Final Output:

- All detected faces have red rectangles, meaning the model classified them as "No Mask."
- Even the face in the painting (Mona Lisa) is detected and marked, maybe Lighting or image quality affecting feature detection.
- Each face detected without a mask is saved as an individual image file.

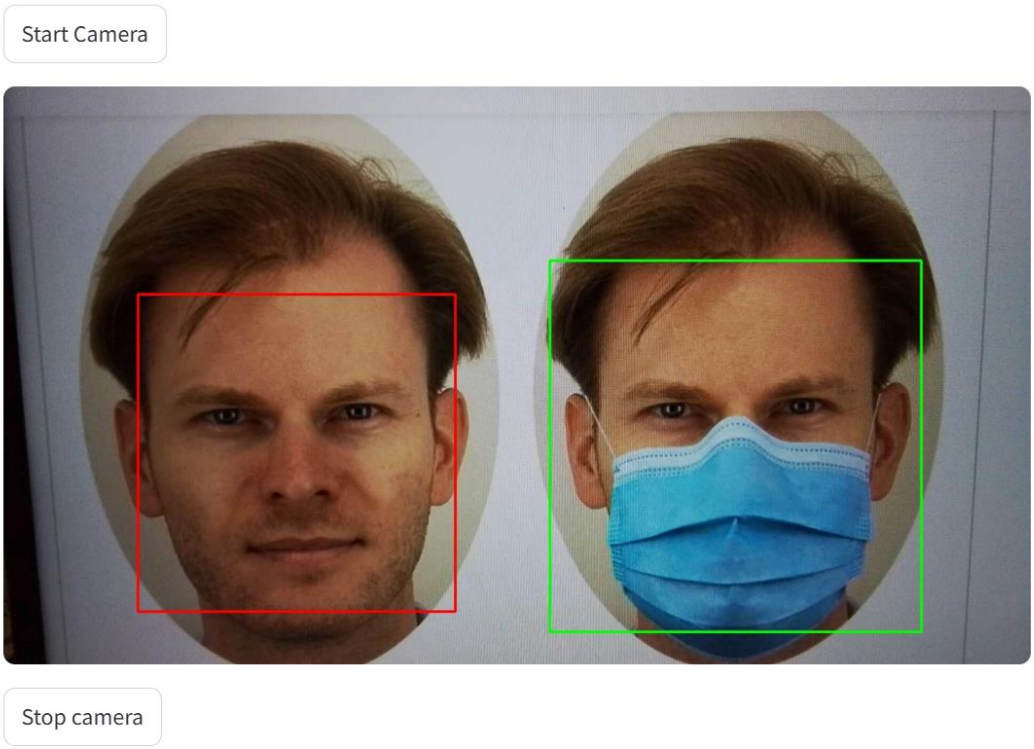
➢ FACE MASK DETECTION FROM AN IP CAMERA USING STREAMLIT

Open the main.py file in your in-code editor, and add the following code for face mask detection from IP Camera, it also saves images of the unmasked faces:

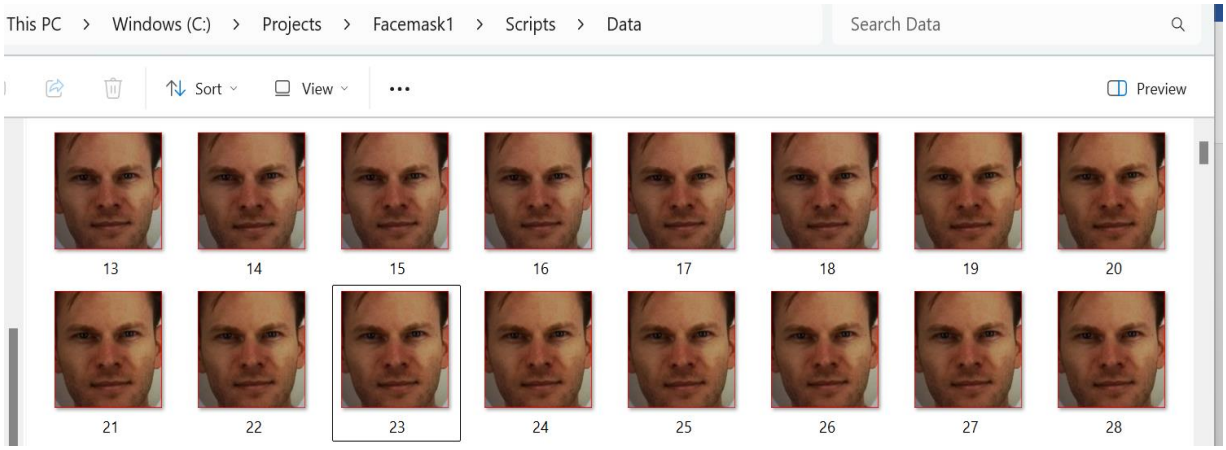
```
elif(choice=="CAMERA"): # Button to start the camera
    btn=st.button("Start Camera")
    window=st.empty()   # Empty container for displaying frames
    btn2=st.button("Stop camera") # Button to stop the camera
    if btn2: # If 'Stop camera' is clicked, rerun the script to stop video capture
        st.rerun()
    if btn: # If 'Start Camera' is clicked
        vid=cv2.VideoCapture("https://192.168.93.205:8080/video")
        i=1  # Counter for saving images
        while(vid.isOpened()):  # Loop until the video is open
            flag, frame = vid.read()# Read a frame from the camera

            if flag:   # Detect faces in the frame

                face=facemodel.detectMultiScale(frame)

                for (x, y, l, w) in face:# Crop the detected face from the frame
                    crop_face1=frame[y:y+w, x:x+l]
                    cv2.imwrite('temp.jpg', crop_face1) # Save the cropped face temporarily

                    # Load and preprocess the cropped face for prediction
                    crop_face=load_img('temp.jpg', target_size=(150, 150, 3))
                    crop_face=img_to_array(crop_face)
                    crop_face=np.expand_dims(crop_face, axis=0)

                    pred=maskmodel.predict(crop_face)[0][0]# Predict mask status (1 = No Mask, 0 = Mask)

                    if pred==1:
                        cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 0, 255), 4) # Draw red rectangle for no mask
                        path = "data/" + str(i) + ".jpg"# Save the face image with no mask in 'data' folder
                        cv2.imwrite(path, crop_face1)
                        i +=1
                    else:
                        cv2.rectangle(frame, (x, y), (x+l, y+w), (0, 255, 0), 4)# Draw green rectangle for mask

                window.image(frame, channels='BGR')  # Display the video frame with rectangles
```

- Two buttons are added using Streamlit – "Start Camera" to begin video capture and "Stop Camera" to stop it.
- An empty container is created in the app layout to continuously display the video frames.
- When the "Stop Camera" button is clicked, the app reruns using st.rerun() to stop the video stream cleanly.
- When "Start Camera" is clicked, the app starts reading video frames from the connected IP camera.
- Each frame is processed, and faces are detected using a pre-trained face detection model like Haar Cascade.
- For every face found, the region is cropped, saved temporarily, and resized to 150x150 for model input.
- The cropped face image is passed to a mask detection model, which predicts whether the person is wearing a mask or not.
- A green rectangle is drawn on faces with masks, and a red rectangle for those without masks – these unmasked face images are also saved in a folder for further use.

Here's a Live steaming in streamlit to detect the face mask:



Here's a folder with unmasked face images:



Final Output:

- The system detects multiple faces in real-time from the webcam feed and highlights them with coloured rectangles.
- The face on the left is marked with a red rectangle, indicating the person is not wearing a mask.
- The face on the right is marked with a green rectangle, showing that the person is correctly wearing a face mask.
- The unmasked person's cropped face image is automatically saved in the "data" folder for record or future processing.

➢ To change the icon and title in Streamlit:

To change the title and icon in Streamlit, first set the title for the browser tab. Then, specify an icon to appear in the tab, which can either be an emoji or an image.

Open the main.py file in your in-code editor, and add the following Streamlit code and put the icon link.

```
st.set_page_config(page_title="Face Mask Detection System", page_icon="https://encrypted-tbn0.gstatic.com/images?q.JPG")
```

This code changed the icon and also the update the title with “Face Mask Detection System”.

SUMMARY OF PROJECT

Summary of Achievement:

We have successfully designed and developed a Face Mask Detection System that meets the specified objectives and integrates essential features. The main highlights of our application include:

- **User-Friendly Interface:** The system features a simple and interactive interface that is easy to operate for users with minimal technical knowledge. It ensures a smooth experience and requires little to no training.
- **Real-Time Face Mask Detection:** The system uses a webcam feed to detect faces and classify them as either wearing a mask or not. It processes live video input and highlights masked faces with green rectangles and unmasked faces with red rectangles.
- **Data Storage for Violators:** The application automatically saves images of individuals not wearing a mask into a dedicated "data" folder, providing a record for monitoring or further review.

Difficulties Encountered During the Project:

- **Learning Curve:** Understanding and implementing image processing, face detection, and mask classification using machine learning models was a challenging task. It required thorough research and hands-on experimentation.
- **Model Integration:** Integrating the face detection model with the mask classification model and ensuring smooth communication between them demanded extensive debugging and adjustment.

Limitations of the Project:

- **Basic Functionalities:** The current system only performs mask detection from webcam input. Advanced features like crowd analysis or multiple camera inputs are yet to be implemented.
- **Limited Accuracy Under Certain Conditions:** The model's accuracy may decrease in poor lighting conditions or when the face is partially visible or covered in unconventional ways.

Future Scope of the Project:

- **Enhanced Detection Models:** Future versions can integrate more advanced AI models to improve detection accuracy and adapt to a wider range of environments.
- **Alert & Notification System:** A real-time alert system can be implemented to notify authorities or concerned parties when an unmasked individual is detected.

By incorporating these enhancements, the Face Mask Detection System can become a more powerful and practical solution for real-time public safety monitoring.

CONCLUSION

The Face Mask Detection System is an intelligent real-time application built using Python, OpenCV, TensorFlow/Keras, and Streamlit to ensure public safety through mask compliance monitoring. It uses a camera feed to detect faces and then classifies whether individuals are wearing a mask or not using a trained deep learning model. The system automatically highlights faces with coloured rectangles: green for masked and red for unmasked individuals: providing instant visual feedback.

A key feature of the system is its ability to save images of unmasked individuals into a designated "data" folder. This helps in maintaining records for future review or compliance checks. The frontend interface is created using Streamlit, offering a user-friendly experience with simple "Start Camera" and "Stop Camera" buttons, live video display, and seamless real-time processing. The system is designed to be lightweight and responsive, suitable for both individual use and deployment in crowded settings like offices or schools.

While the system performs its main task effectively, there is strong potential for future improvements. Enhancements such as real-time SMS or email alerts for unmasked detection, cloud-based storage for centralized monitoring, and integration with identity recognition could make it more powerful. These additions would increase the system’s reliability and widen its application scope, making it an essential tool for health and safety monitoring in public and private spaces.