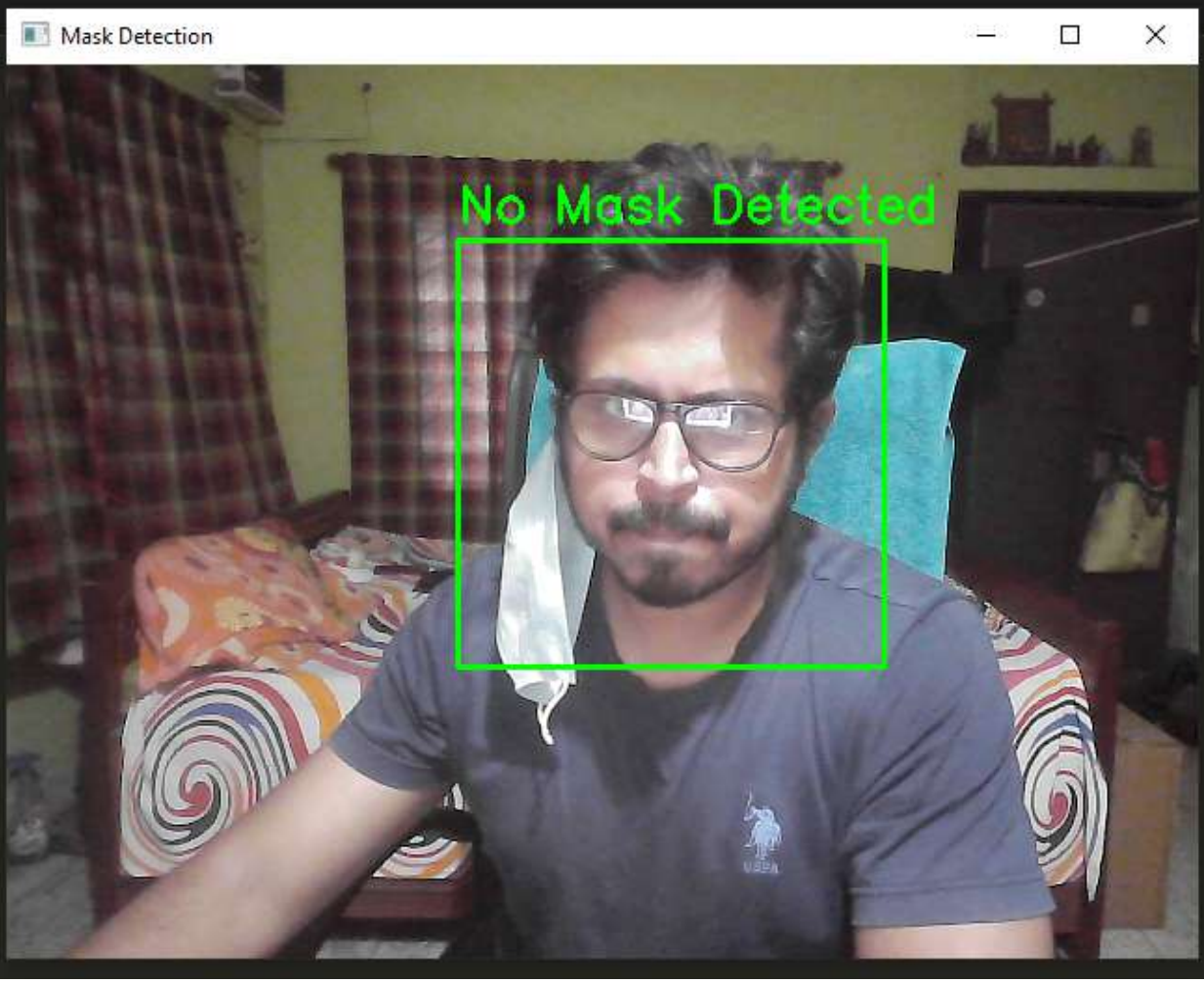

Mask Detection
Mask Detected


Mask Detection
No Mask Detected

```python
import zipfile
import subprocess

import kaggle

command = 'kaggle datasets download -d omkargurav/face-mask-dataset'

subprocess.run(command,shell=True)

zip_path = 'face-mask-dataset.zip'

with zipfile.ZipFile(zip_path,'r') as zip_ref:
    zip_ref.extractall('D:\Face_mask_detect')
```

```python
import os
import numpy as np
from keras.preprocessing import image
import cv2
import warnings
warnings.filterwarnings('ignore')
import random


def get_XY():
    categories = ['with_mask', 'without_mask']
    dataset = []

    for category in categories:
        path = os.path.join('data', category)
        label = categories.index(category)
        file_list = os.listdir(path)
        random.shuffle(file_list)  # Shuffle the file list

        count = 0  # Track the number of selected photos
        for file in file_list:
            if count >= 1200:
                break  # Stop iterating if we have reached the desired count

            img_path = os.path.join(path, file)
            img = cv2.imread(img_path)
            img = cv2.resize(img, (224, 224))

            dataset.append([img, label])
            count += 1

    random.shuffle(dataset)

    X = []
    Y = []

    for features, label in dataset:
        X.append(features)
        Y.append(label)

    X = np.array(X)
    X = X / 255  # scaling the X
    Y = np.array(Y)

    return X, Y
```

```
import tensorflow
from tensorflow import keras


def create_model():

    num_of_classes = 2

    model = keras.Sequential()

    try:

        model.add(keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(224,224,3)))
        model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))


        model.add(keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'))
        model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

        model.add(keras.layers.Flatten())

        model.add(keras.layers.Dense(128, activation='relu'))
        model.add(keras.layers.Dropout(0.5))

        model.add(keras.layers.Dense(64, activation='relu'))
        model.add(keras.layers.Dropout(0.5))


        model.add(keras.layers.Dense(num_of_classes, activation='sigmoid'))

        return model

    except ValueError as ve:
        print("Value Error occured:",str(ve))
    except TypeError as te:
        print("TypeError occured:",str(te))
    except Exception as e:
        print("An error occured:",str(e))

create_model()
```

```python
from sklearn.model_selection import train_test_split
from data_prep import get_XY
from model import create_model
import pickle


#load the data

X,Y = get_XY()


#split the data

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

model = create_model()

# compile the neural network
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
history = model.fit(X_train, Y_train, validation_split=0.1, epochs=7)
model.save('Trained_model.h5')

with open("training_history.pkl",'wb') as f:
    pickle.dump(history.history,f)
loss, accuracy = model.evaluate(X_test, Y_test)
print('Test Accuracy =', accuracy*100,'%')
```

```python
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np


def predict_mask(input_image_path, model):
    input_image = cv2.imread(input_image_path)
    input_image_resized = cv2.resize(input_image, (224, 224))
    input_image_scaled = input_image_resized / 255
    input_image_reshaped = np.reshape(input_image_scaled, [1, 224, 224, 3])
    input_prediction = model.predict(input_image_reshaped)
    input_pred_label = np.argmax(input_prediction)

    plt.imshow(cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

    if input_pred_label == 0:
        print('The person in the image is wearing a mask')
    else:
        print('The person in the image is not wearing a mask')

# Usage example
input_image_path = input('Path of the image to be predicted: ')
model_path = 'D:\Face_mask_detect\Trained_model.h5'  # Replace with the actual path to your trained model

model = tf.keras.models.load_model(model_path)

predict_mask(input_image_path, model)
```

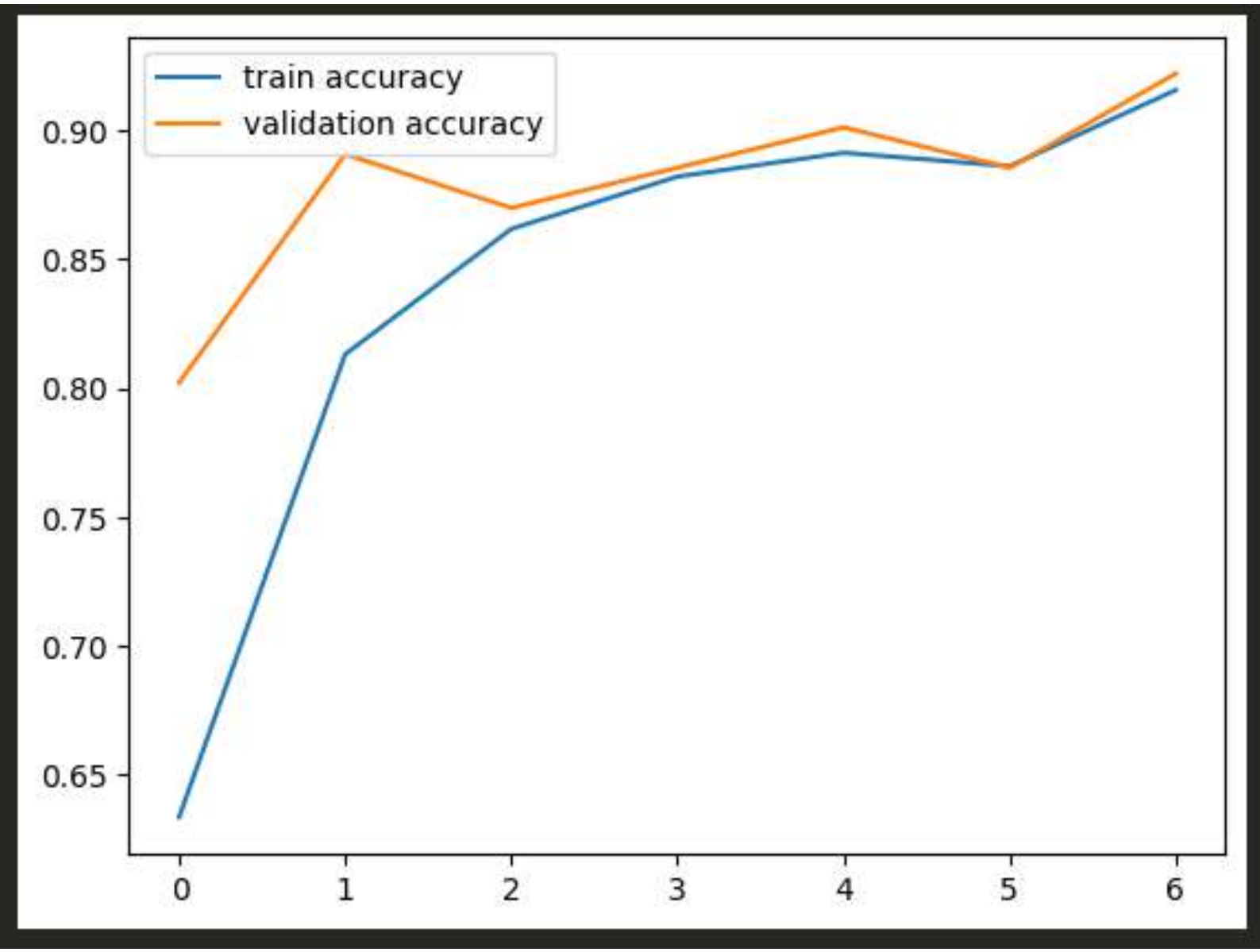
```python
# Load the pre-trained model
model_path = 'E:\GITHUBREpo\Deep_learning-Neural-Network-projects-\Trained_model.h5'  # Replace with the actual path to your trained model
model = tf.keras.models.load_model(model_path)

# Load the face cascade classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Function to perform mask prediction on the image
def predict_mask(image):
    # Resize and preprocess the image
    input_image_resized = cv2.resize(image, (224, 224))
    input_image_scaled = input_image_resized / 255
    input_image_reshaped = np.reshape(input_image_scaled, [1, 224, 224, 3])

    # Perform mask prediction
    input_prediction = model.predict(input_image_reshaped)
    input_pred_label = np.argmax(input_prediction)

    if input_pred_label == 0:
        return 'Mask Detected'
    else:
        return 'No Mask Detected'

# Open the webcam
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Failed to open webcam.")
    exit()

# Create a pop-up window to display the webcam feed
cv2.namedWindow('Mask Detection')

# Process frames from the webcam feed
while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    if not ret:
        print("Failed to capture frame from webcam.")
        break

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

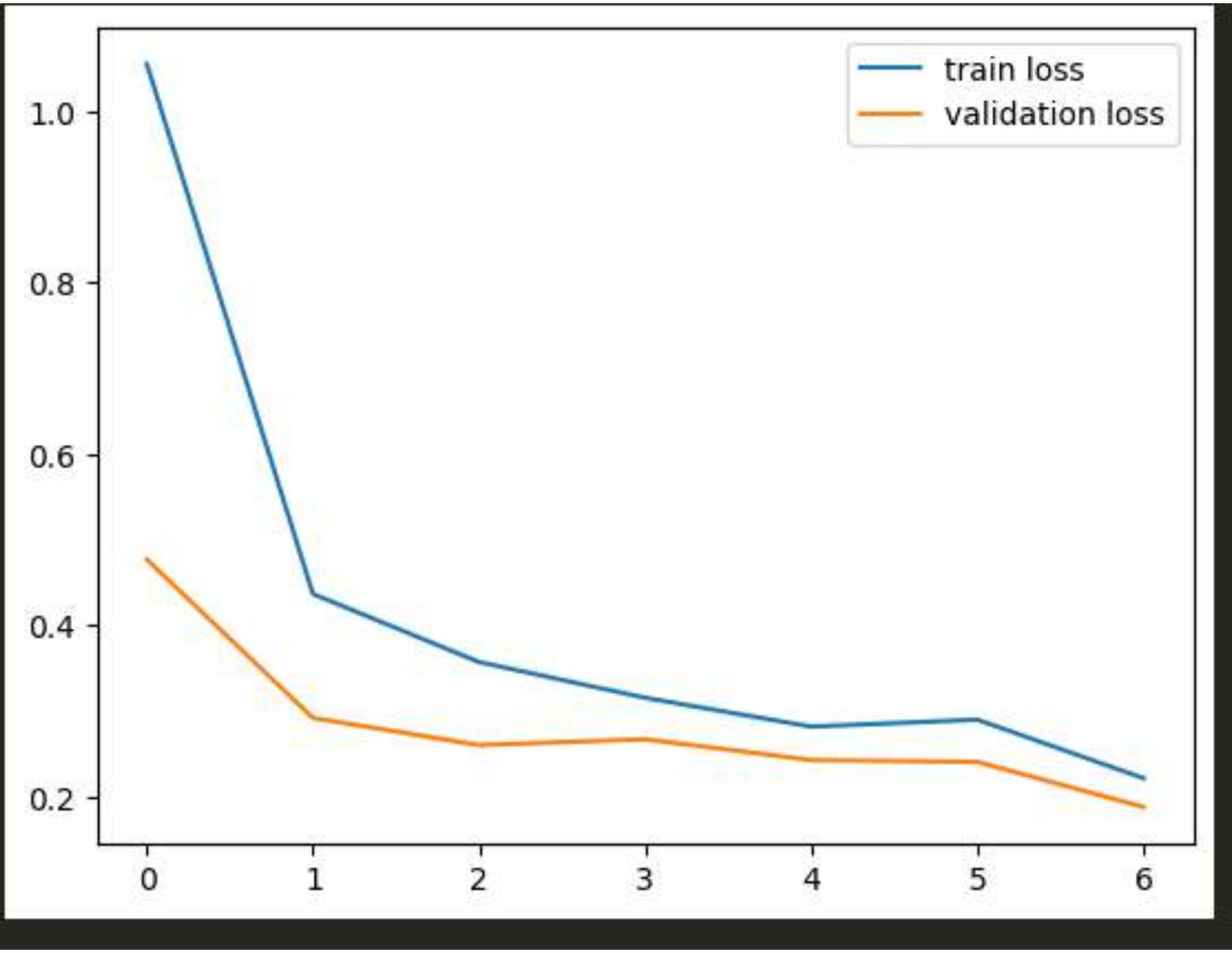
```
    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Find the largest face
    largest_face = None
    largest_area = 0

    for (x, y, w, h) in faces:
        area = w * h
        if area > largest_area:
            largest_area = area
            largest_face = (x, y, w, h)

    # Process the largest face
    if largest_face is not None:
        x, y, w, h = largest_face

        # Zoom in on the face by adjusting the region of interest (ROI)
        roi_x = x - int(0.2 * w)
        roi_y = y - int(0.2 * h)
        roi_w = int(1.4 * w)
        roi_h = int(1.4 * h)
        roi = frame[roi_y:roi_y + roi_h, roi_x:roi_x + roi_w]

        if roi.size != 0:  # Check if ROI is empty
            try:
                # Perform mask prediction on the ROI
                result = predict_mask(roi)

                # Draw bounding box and label on the face
                cv2.rectangle(frame, (roi_x, roi_y), (roi_x + roi_w, roi_y + roi_h), (0, 255, 0), 2)
                cv2.putText(frame, result, (roi_x, roi_y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
            except Exception as e:
                print("Error during prediction:", str(e))

    # Display the frame in the pop-up window
    cv2.imshow('Mask Detection', frame)

    # Break the loop when 'x' is pressed
    if cv2.waitKey(1) & 0xFF == ord('x'):
        break

# Release the resources
cap.release()
cv2.destroyAllWindows()
```

```
1/1 [==============================] - 0s 78ms/step
```

```
The person in the image is wearing a mask
```

The person in the image is not wearing a mask


train accuracy
validation accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0
1
2
3
4
5
6


train loss
validation loss
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6