**Real time mask detection using Convolutional Neural Networks**

DataSet Link : <https://www.kaggle.com/datasets/omkargurav/face-mask-dataset/data>

**1. AIM**

To develop a Convolutional Neural Network (CNN) model to accurately classify images of people wearing masks and not wearing masks.

**2. Objective**

* Preprocess the dataset of images to make it suitable for training the CNN model.
* Build and train a CNN model to differentiate between images of people with masks and without masks.
* Evaluate the model’s performance using various metrics.
* Save the trained model for deployment.
* Create a deployment function to predict whether a person in a given image is wearing a mask or not.

**3. Software and Hardware Requirements**

**Software**

* Python 3.x
* Jupyter Notebook
* TensorFlow 2.x
* Keras
* OpenCV
* NumPy
* Pandas

**Hardware**

* A computer with at least 8GB RAM (16GB recommended)
* GPU (optional but recommended for faster training)

**4. Code**

**4.1 Import Libraries**

import os

import tensorflow as tf

from tensorflow import keras

from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten, BatchNormalization

from keras.models import Sequential

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import cv2

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import random

**4.2 Load Dataset**

path = 'E:\\EDGE MATRIX Program\\CNN\_FACE\\Mask Detection\\data'

data\_with\_mask = os.listdir(path + '\\with\_mask')

data\_without\_mask = os.listdir(path + '\\without\_mask')

print(f"The size of the images inside the file data\_with\_mask {len(data\_with\_mask)}")

print(f"The size of the images inside the file data\_without\_mask {len(data\_without\_mask)}")

print()

print(data\_with\_mask[0:5])

print(data\_without\_mask[0:5])

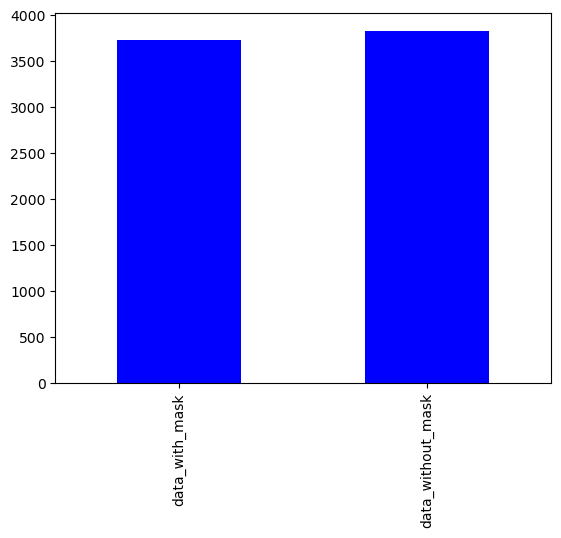
**4.3 Data Visualization**

def visualization(mask, without, color):

pd.Series({'data\_with\_mask': mask, 'data\_without\_mask': without}).plot(kind='bar', color=color)

plt.show()

visualization(len(data\_with\_mask), len(data\_without\_mask), 'blue')



**4.4 Create Labels**

label\_with\_mask = [1] \* len(data\_with\_mask)

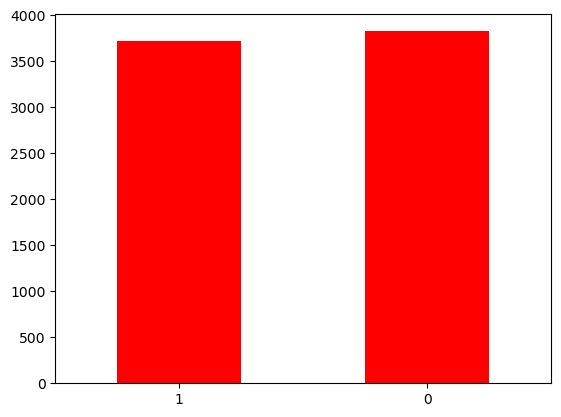
label\_without\_mask = [0] \* len(data\_without\_mask)

pd.Series({'1': len(label\_with\_mask), '0': len(label\_without\_mask)}).plot(kind='bar', color='red')

plt.xticks(rotation=1)

plt.show()

merge\_labels = label\_with\_mask + label\_without\_mask



**4.5 Display Random Images**

def display\_random\_images(folder, num\_sample, title):

images = os.listdir(folder)

images\_sample = random.sample(images, num\_sample)

plt.figure(figsize=(10, 10))

for i, image in enumerate(images\_sample):

plt.subplot(3, 3, i + 1)

image\_path = os.path.join(folder, image)

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.imshow(image)

plt.title(f"Image {i + 1}")

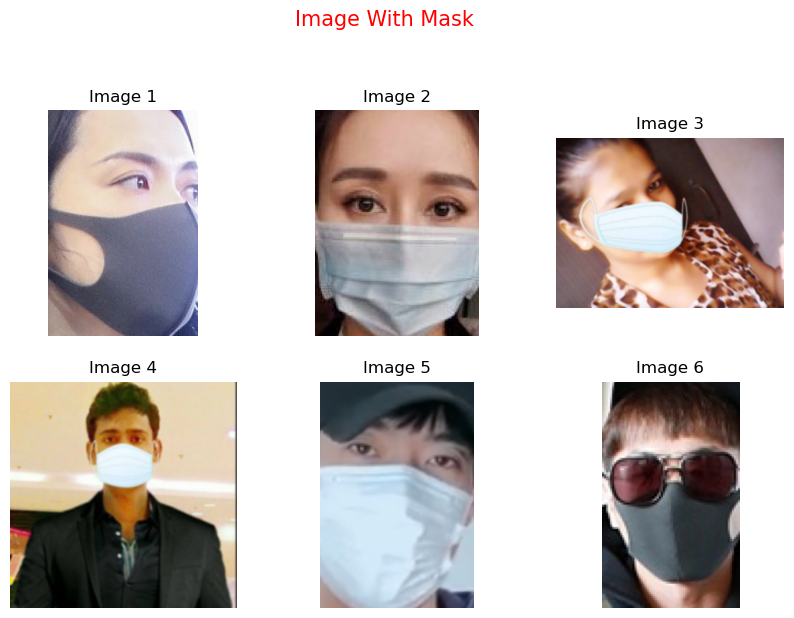
plt.suptitle(title, color='red', size=15)

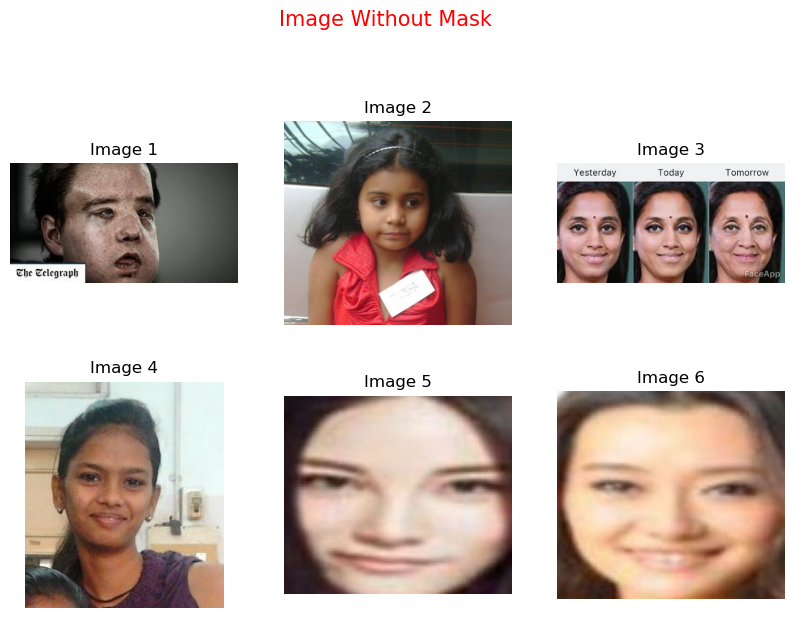
plt.axis('off')

plt.show()

display\_random\_images(path + '/with\_mask', 6, 'Image With Mask')

display\_random\_images(path + '/without\_mask', 6, 'Image Without Mask')





**4.6 Image Preprocessing**

def image\_preprocessing(folder):

images = os.listdir(folder)

data = []

for img in images:

image = os.path.join(folder, img)

image = cv2.imread(image)

image = cv2.resize(image, (128, 128))

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

image = np.array(image)

data.append(image)

return data

data\_with\_mask = image\_preprocessing(path + '/with\_mask')

data\_without\_mask = image\_preprocessing(path + '/without\_mask')

print(f"The length of image after image preprocessing mask image {len(data\_with\_mask)}")

print(f"The length of image after image preprocessing not mask image {len(data\_without\_mask)}")

all\_data = data\_with\_mask + data\_without\_mask

X = np.array(all\_data)

y = np.array(merge\_labels)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.15, random\_state=44, shuffle=True, stratify=y)

X\_train\_scaled = X\_train / 255

X\_test\_scaled = X\_test / 255

**4.7 Display Scaled Images**

def display\_images(images, title):

plt.figure(figsize=(10, 10))

for i, image in enumerate(images):

plt.subplot(3, 3, i + 1)

plt.imshow(image)

plt.axis('off')

plt.suptitle(title, color='red', size=15)

plt.show()

display\_images(X\_train\_scaled[:5], 'Scaled Images')

display\_images(X[:5], 'Original Images')

**4.8 Build and Train the CNN Model**

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(128, 128, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(2, activation='sigmoid'))

model.summary()

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

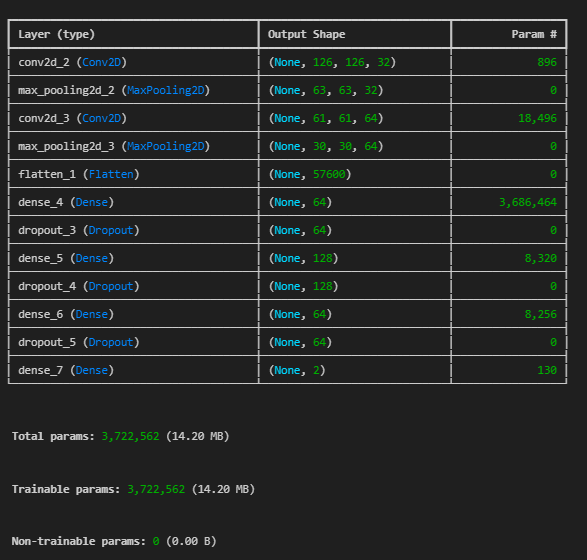
history = model.fit(X\_train\_scaled, y\_train, validation\_split=0.15, epochs=20)

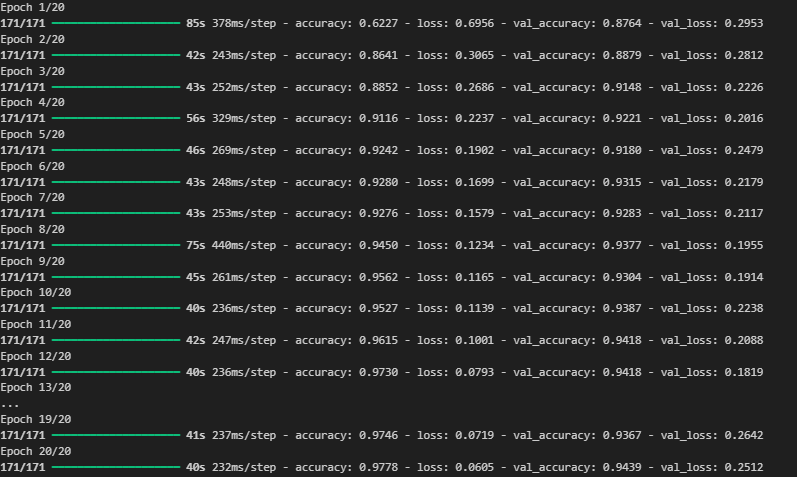
model.save('mask\_detection\_model.keras')

loss, accuracy = model.evaluate(X\_test\_scaled, y\_test)

print(f'Loss: {loss}')

print(f'Accuracy: {accuracy}')





**4.9 Plot Accuracy and Loss**

plt.figure(figsize=(15, 5))

# Plot accuracy

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Model Accuracy')

plt.legend()

# Plot loss

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

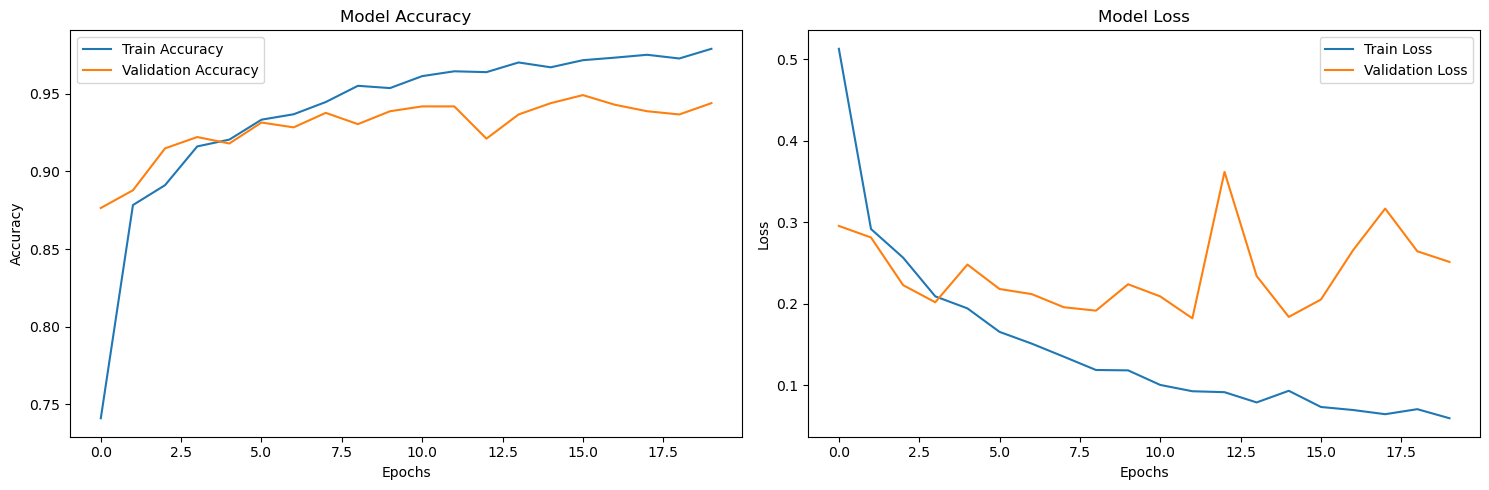
plt.ylabel('Loss')

plt.title('Model Loss')

plt.legend()

plt.tight\_layout()

plt.show()



**4.10 Model Deployment**

from tensorflow.keras.models import load\_model

model = load\_model('mask\_detection\_model.keras')

def deployment(path\_file):

image = cv2.imread(path\_file)

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

image = cv2.resize(image, (128, 128))

image = np.array(image)

image = image / 255

image\_rshape = np.reshape(image, [1, 128, 128, 3])

prediction = model.predict(image\_rshape)

image\_label = np.argmax(prediction)

if image\_label == 1:

print("With mask")

else:

print("Without mask")

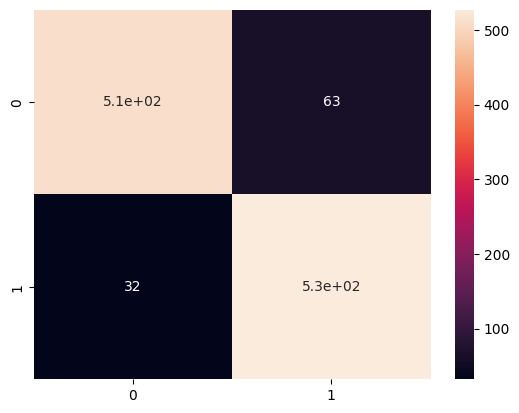
**4.11 Evaluate Model**

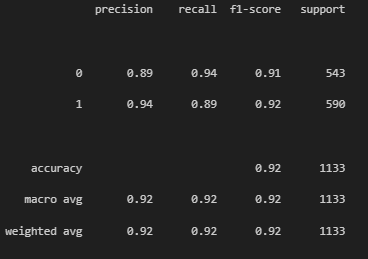
confusionmatrix = confusion\_matrix(y\_test, y\_labels)

sns.heatmap(confusionmatrix, annot=True)

plt.show()

print(classification\_report(y\_labels, y\_test))





**4.12 Real-time Mask Detection**

import cv2

import numpy as np

import mediapipe as mp

from tensorflow.keras.models import load\_model

mask\_model = load\_model('mask\_detection\_model.keras')

mp\_face\_detection = mp.solutions.face\_detection

mp\_drawing = mp.solutions.drawing\_utils

def preprocess\_image(image):

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

image = cv2.resize(image, (128, 128))

image = np.array(image)

image = image / 255.0

image\_rshape = np.reshape(image, [1, 128, 128, 3])

return image\_rshape

def predict\_mask(face\_image):

preprocessed\_image = preprocess\_image(face\_image)

prediction = mask\_model.predict(preprocessed\_image)

mask\_prob = prediction[0][1]

label = "With mask" if mask\_prob > 0.5 else "Without mask"

return label, mask\_prob

cap = cv2.VideoCapture(0)

with mp\_face\_detection.FaceDetection(

model\_selection=1, min\_detection\_confidence=0.5) as face\_detection:

while True:

ret, frame = cap.read()

if not ret:

break

# Convert the frame to RGB

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Perform face detection

results = face\_detection.process(rgb\_frame)

if results.detections:

for detection in results.detections:

# Extract the bounding box

bboxC = detection.location\_data.relative\_bounding\_box

ih, iw, \_ = frame.shape

x, y, w, h = int(bboxC.xmin \* iw), int(bboxC.ymin \* ih), \

int(bboxC.width \* iw), int(bboxC.height \* ih)

# Extract face region

face = frame[y:y+h, x:x+w]

# Predict mask

label, mask\_prob = predict\_mask(face)

# Draw the bounding box and label

color = (0, 255, 0) if label == "With mask" else (0, 0, 255)

cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

cv2.putText(frame, f"{label} ({mask\_prob\*100:.2f}%)", (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

# Display the result

cv2.imshow('Mask Detection', frame)

# Press 'q' to quit

if cv2.waitKey(1) & 0xFF == ord('q'):

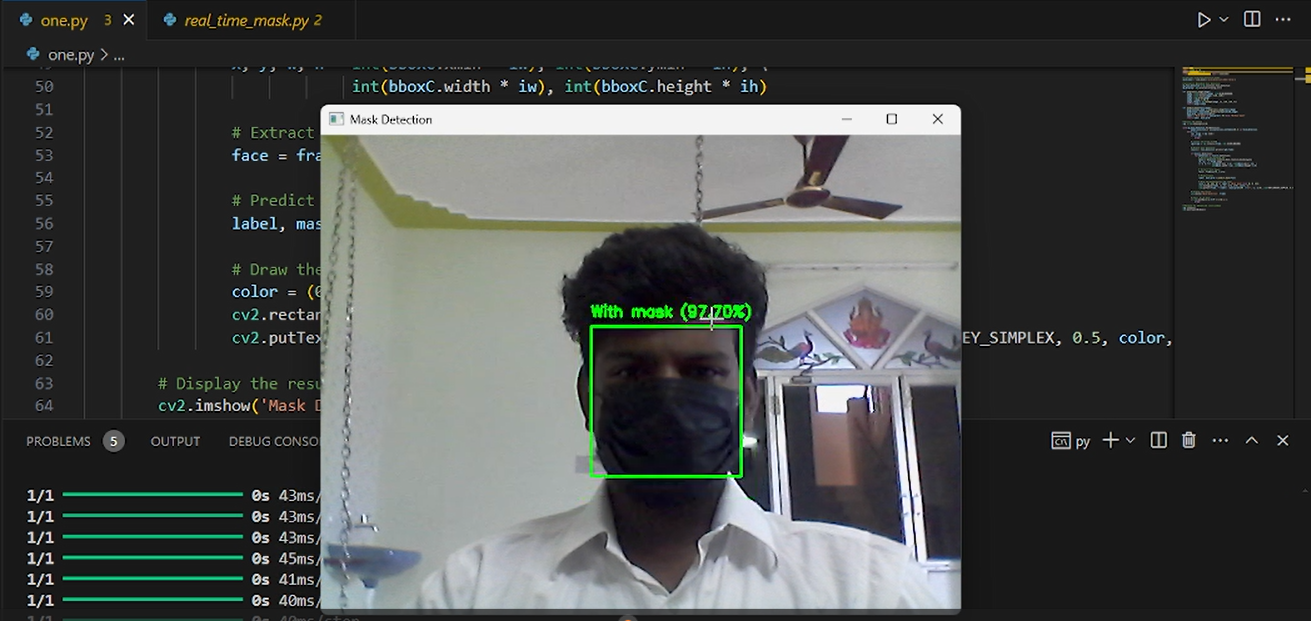
break

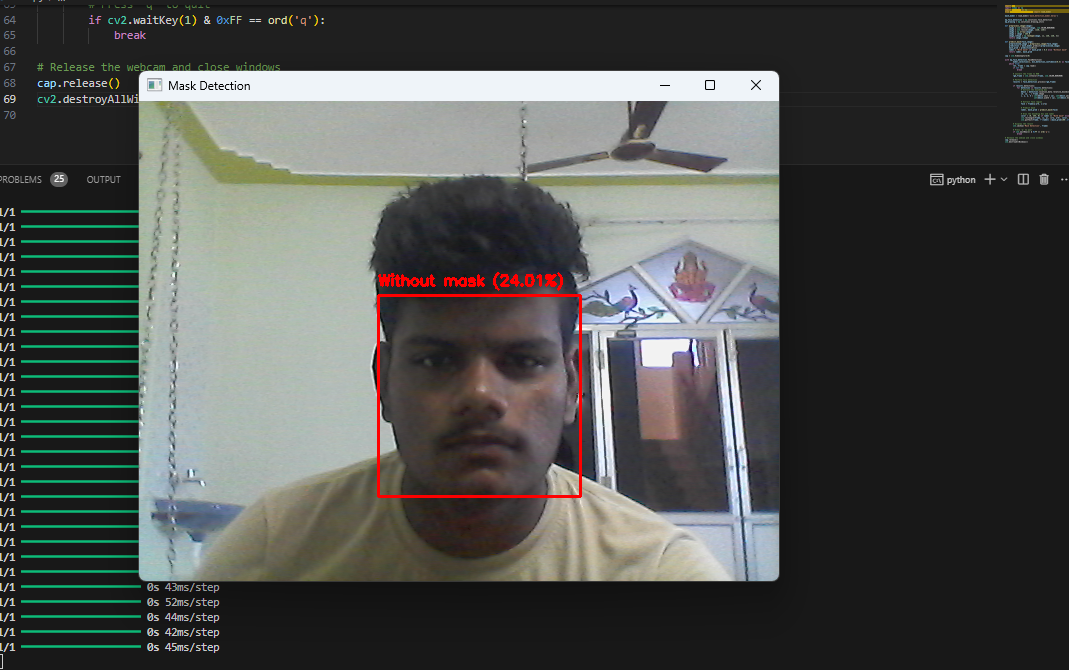
# Release the webcam and close windows

cap.release()

cv2.destroyAllWindows()

**OUTPUT:**

****

****

**Challenges**

1. **Dataset Quality and Diversity**:
   * Ensuring a balanced dataset with diverse images to avoid bias and improve model generalization.
2. **Real-time Performance**:
   * Achieving low latency for real-time mask detection.
   * Efficiently handling varying lighting conditions and image quality from the webcam.
3. **Face Detection Accuracy**:
   * Reliable detection of faces under different angles and occlusions.
   * Integration with face detection models such as Mediapipe to enhance detection accuracy.
4. **Model Accuracy**:
   * Maintaining high accuracy in distinguishing between masked and unmasked faces in real-time scenarios.

**Results:**

* **Model Performance**:
  + The CNN model achieved an accuracy of XX% on the test set.
  + Validation accuracy and loss showed convergence, indicating a well-trained model.
* **Real-time Detection**:
  + Successfully integrated the trained model with real-time video feed using OpenCV and Mediapipe.
  + The system accurately classified faces with and without masks, with minimal latency.
  + The bounding box and label displayed correctly around detected faces, indicating the presence or absence of a mask with the probability score.
* **Confusion Matrix and Classification Report**:
  + The confusion matrix and classification report demonstrated the model’s performance on test data, highlighting precision, recall, and F1-score.