Real time mask detection using Convolutional Neural Networks

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
- Matplotlib
- Seaborn
- Scikit-learn

Hardware

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

4. Code

4.1 Import Libraries

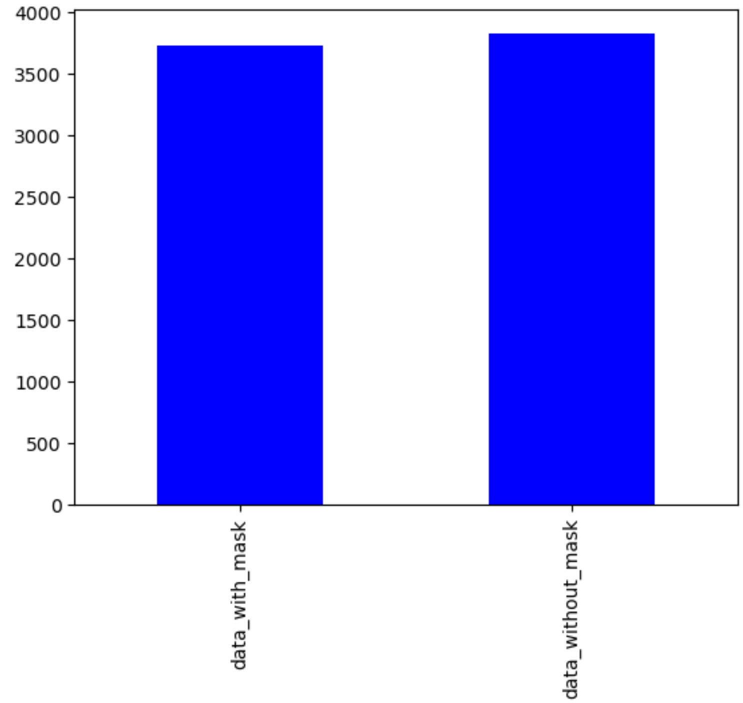
```
import os

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])
```

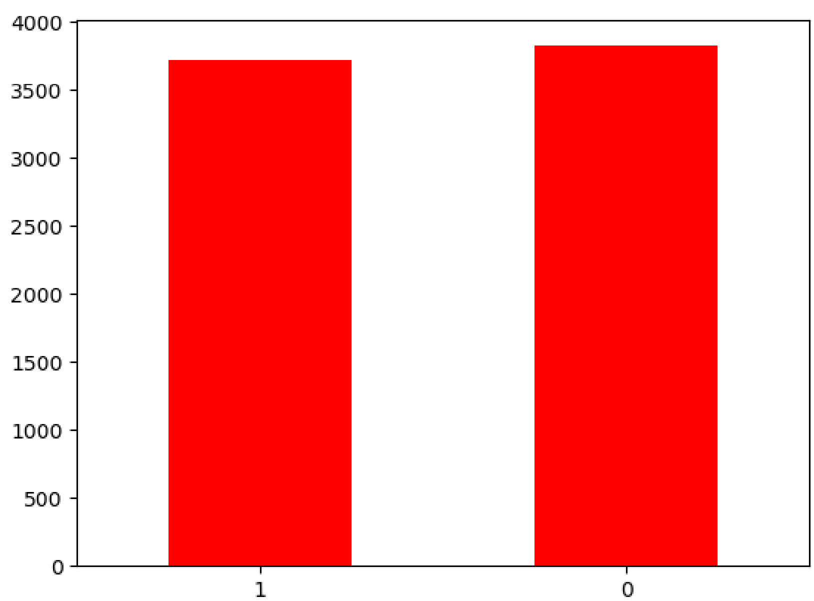
4.3 Data Visualization

```
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')
```



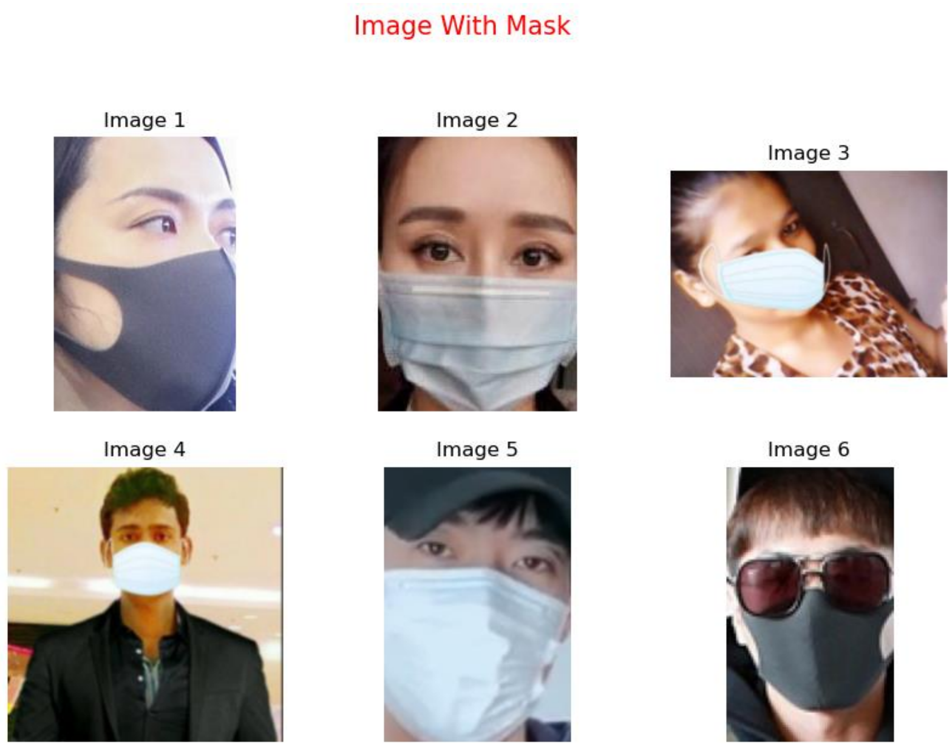
4.4 Create Labels

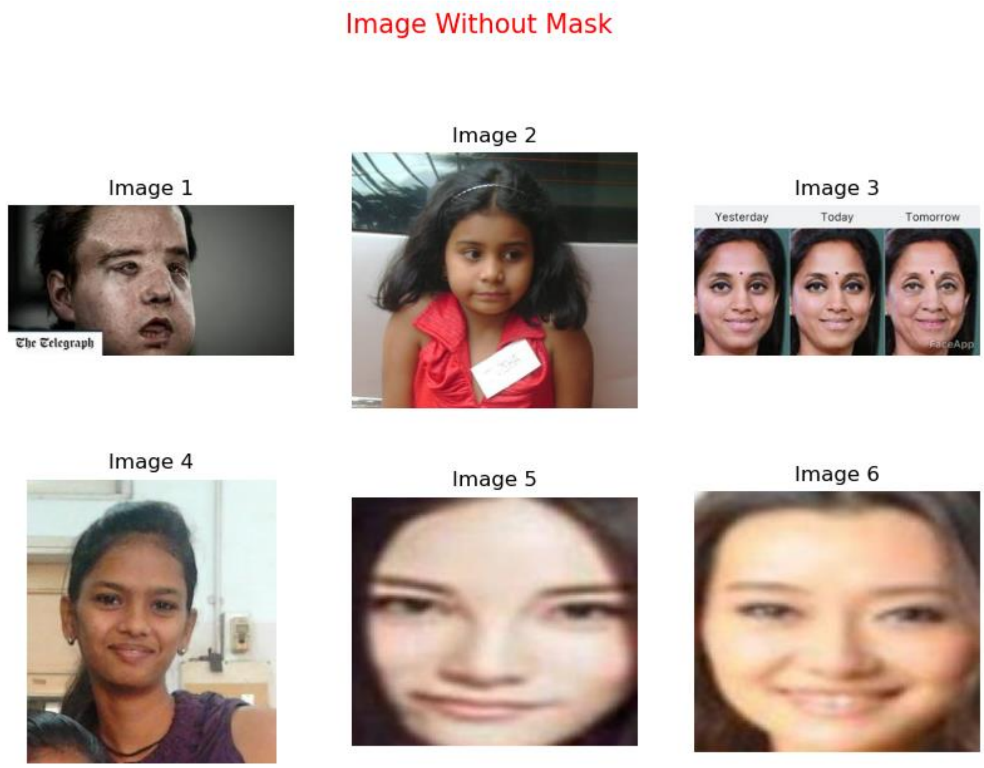
```
label_with_mask = [1] * len(data_with_mask)
label_without_mask = [0] * len(data_without_mask)

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}')
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_3 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten_1 (Flatten)	(None, 57600)	0
dense_4 (Dense)	(None, 64)	3,686,464
dropout_3 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 128)	8,320
dropout_4 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8,256
dropout_5 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 2)	130

Total params: 3,722,562 (14.20 MB)

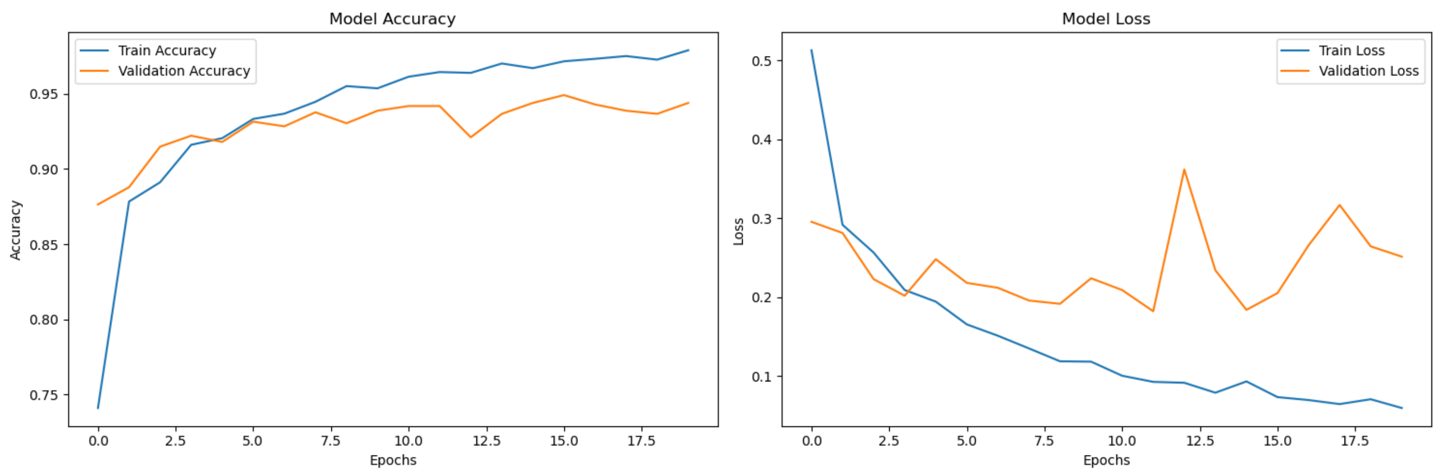
Trainable params: 3,722,562 (14.20 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 85s 378ms/step - accuracy: 0.6227 - loss: 0.6956 - val_accuracy: 0.8764 - val_loss: 0.2953
Epoch 2/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 42s 243ms/step - accuracy: 0.8641 - loss: 0.3065 - val_accuracy: 0.8879 - val_loss: 0.2812
Epoch 3/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 43s 252ms/step - accuracy: 0.8852 - loss: 0.2686 - val_accuracy: 0.9148 - val_loss: 0.2226
Epoch 4/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 56s 329ms/step - accuracy: 0.9116 - loss: 0.2237 - val_accuracy: 0.9221 - val_loss: 0.2016
Epoch 5/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 46s 269ms/step - accuracy: 0.9242 - loss: 0.1902 - val_accuracy: 0.9180 - val_loss: 0.2479
Epoch 6/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 43s 248ms/step - accuracy: 0.9280 - loss: 0.1699 - val_accuracy: 0.9315 - val_loss: 0.2179
Epoch 7/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 43s 253ms/step - accuracy: 0.9276 - loss: 0.1579 - val_accuracy: 0.9283 - val_loss: 0.2117
Epoch 8/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 75s 440ms/step - accuracy: 0.9450 - loss: 0.1234 - val_accuracy: 0.9377 - val_loss: 0.1955
Epoch 9/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 45s 261ms/step - accuracy: 0.9562 - loss: 0.1165 - val_accuracy: 0.9304 - val_loss: 0.1914
Epoch 10/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 40s 236ms/step - accuracy: 0.9527 - loss: 0.1139 - val_accuracy: 0.9387 - val_loss: 0.2238
Epoch 11/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 42s 247ms/step - accuracy: 0.9615 - loss: 0.1001 - val_accuracy: 0.9418 - val_loss: 0.2088
Epoch 12/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 40s 236ms/step - accuracy: 0.9730 - loss: 0.0793 - val_accuracy: 0.9418 - val_loss: 0.1819
Epoch 13/20
...
Epoch 19/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 41s 237ms/step - accuracy: 0.9746 - loss: 0.0719 - val_accuracy: 0.9367 - val_loss: 0.2642
Epoch 20/20
171/171 ━━━━━━━━━━━━━━━━━━━━ 40s 232ms/step - accuracy: 0.9778 - loss: 0.0605 - val_accuracy: 0.9439 - val_loss: 0.2512
```

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()
```



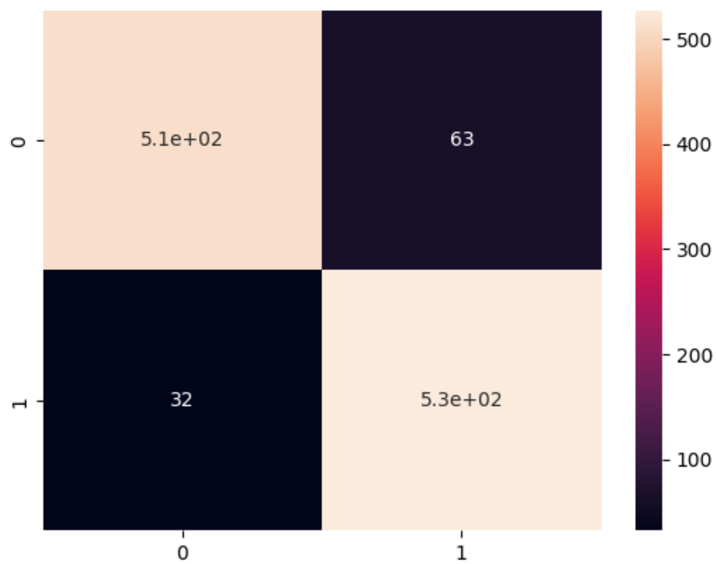
4.10 Model Deployment

```
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))
```



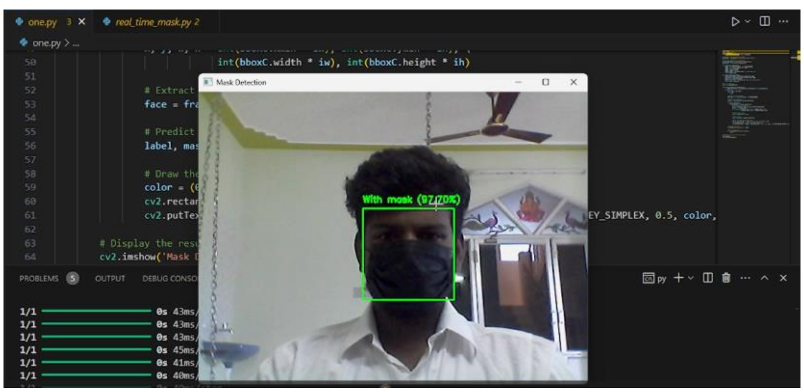

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       543
           1       0.94      0.89      0.92       590

    accuracy                           0.92      1133
   macro avg       0.92      0.92      0.92      1133
weighted avg       0.92      0.92      0.92      1133
```

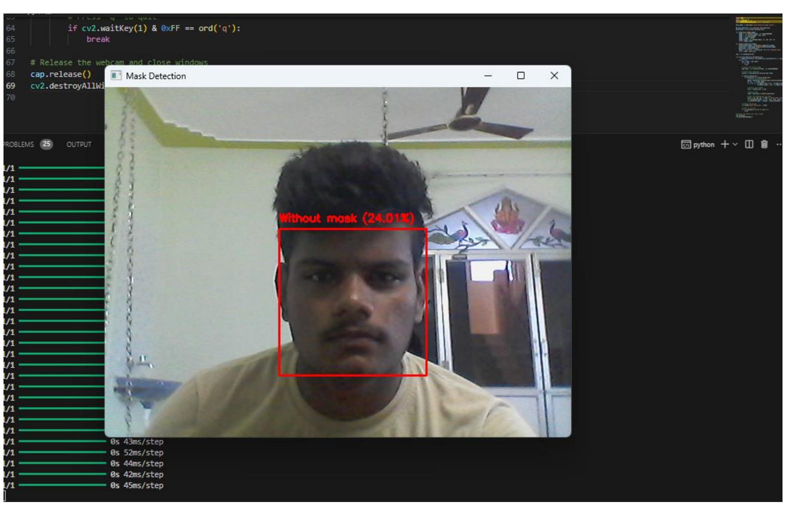
4.12 Real-time Mask Detection

```
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.