**Importing the Dependencies**

import os  
import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
from flask import Flask, request, render\_template, render\_template\_string  
from PIL import Image  
from sklearn.model\_selection import train\_test\_split  
import tensorflow as tf  
import streamlit as st  
from tensorflow import keras

**Create Flask app**

app = Flask(\_\_name\_\_)

**Prepare data**

Real\_File = os.listdir('D:\deepfake\_detector\Data\Real')  
print(Real\_File[0:5])  
print(Real\_File[-5:])

['Real (1).jpg', 'Real (10).jpg', 'Real (100).jpg', 'Real (1000).jpg', 'Real (1001).jpg']  
['Real (995).jpg', 'Real (996).jpg', 'Real (997).jpg', 'Real (998).jpg', 'Real (999).jpg']

Fake\_File = os.listdir('D:\deepfake\_detector\Data\Fake')  
print(Fake\_File[0:5])  
print(Fake\_File[-5:])

['Fake (1).jpg', 'Fake (10).jpg', 'Fake (100).jpg', 'Fake (1000).jpg', 'Fake (1001).jpg']  
['Fake (995).jpg', 'Fake (996).jpg', 'Fake (997).jpg', 'Fake (998).jpg', 'Fake (999).jpg']

print('Number of real images:', len(Real\_File))  
print('Number of fake images:', len(Fake\_File))

Number of real images: 2000  
Number of fake images: 2000

**Creating Labels for the two class of Images**

Real Image --> 1

Fake Image --> 0

**1.Load data**

**2.Image processing**

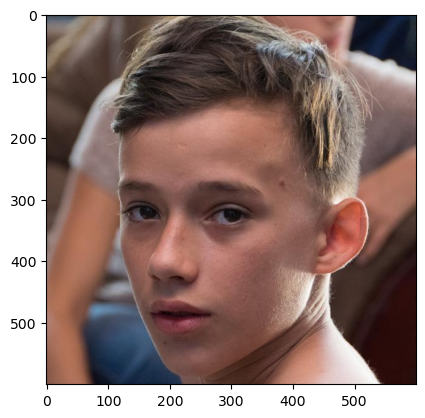
a. Resize the Images

b. Convert the images to numpy arrays+

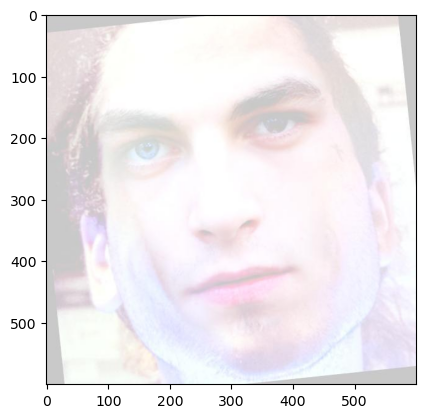
real\_path='D:\deepfake\_detector\Data\Real'  
fake\_path='D:\deepfake\_detector\Data\Fake'  
  
def load\_data(real\_path, fake\_path):  
 real\_files = os.listdir(real\_path)  
 fake\_files = os.listdir(fake\_path)  
  
 data = []  
 labels = []  
  
 for img\_file in real\_files:  
 image = Image.open(os.path.join(real\_path, img\_file)).resize((128, 128)).convert('RGB')  
 data.append(np.array(image))  
 labels.append(1) # Real images  
  
 for img\_file in fake\_files:  
 image = Image.open(os.path.join(fake\_path, img\_file)).resize((128, 128)).convert('RGB')  
 data.append(np.array(image))  
 labels.append(0) # Fake images  
   
 return np.array(data), np.array(labels)

**Displaying the Images**

# displaying with real image  
img = mpimg.imread('D:\deepfake\_detector\Data\Real\Real (1).jpg')  
imgplot = plt.imshow(img)  
plt.show()



# displaying fake image  
img = mpimg.imread('D:\deepfake\_detector\Data\Fake\Fake (1).jpg')  
imgplot = plt.imshow(img)  
plt.show()



**Load the dataset**

X, Y = load\_data('D:\deepfake\_detector\Data\Real', 'D:\deepfake\_detector\Data\Fake')  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

**Create and train the model**

# Building a Convolutional Neural Networks (CNN)  
def create\_model():  
 model = keras.Sequential([  
 keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(128, 128, 3)),  
 keras.layers.MaxPooling2D(pool\_size=(2, 2)),  
 keras.layers.Conv2D(64, kernel\_size=(3, 3), activation='relu'),  
 keras.layers.MaxPooling2D(pool\_size=(2, 2)),  
 keras.layers.Flatten(),  
 keras.layers.Dense(128, activation='relu'),  
 keras.layers.Dropout(0.5),  
 keras.layers.Dense(1, activation='sigmoid')  
 ])  
 # compile the neural network  
 model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
 return model  
  
model = create\_model()  
  
# training the neural network  
history=model.fit(X\_train / 255.0, Y\_train, validation\_data=(X\_test / 255.0, Y\_test), epochs=10, batch\_size=32)

c:\Users\asmaa\anaconda3\envs\project\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
 super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

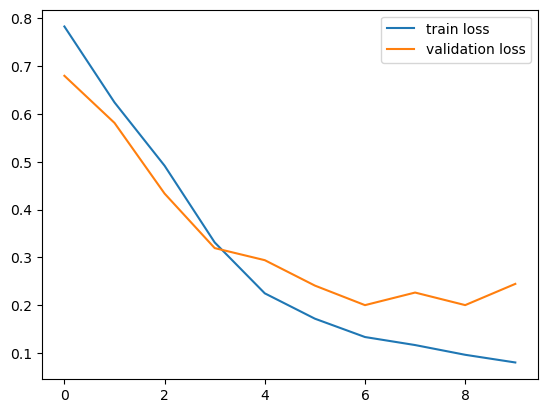
Epoch 1/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 32s 293ms/step - accuracy: 0.5007 - loss: 1.0359 - val\_accuracy: 0.6313 - val\_loss: 0.6797  
Epoch 2/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 269ms/step - accuracy: 0.6338 - loss: 0.6488 - val\_accuracy: 0.6888 - val\_loss: 0.5812  
Epoch 3/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 244ms/step - accuracy: 0.7412 - loss: 0.5151 - val\_accuracy: 0.8263 - val\_loss: 0.4335  
Epoch 4/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 245ms/step - accuracy: 0.8384 - loss: 0.3508 - val\_accuracy: 0.8625 - val\_loss: 0.3195  
Epoch 5/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 242ms/step - accuracy: 0.8933 - loss: 0.2407 - val\_accuracy: 0.8650 - val\_loss: 0.2942  
Epoch 6/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 247ms/step - accuracy: 0.9282 - loss: 0.1783 - val\_accuracy: 0.8963 - val\_loss: 0.2411  
Epoch 7/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 245ms/step - accuracy: 0.9414 - loss: 0.1343 - val\_accuracy: 0.9087 - val\_loss: 0.2002  
Epoch 8/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 245ms/step - accuracy: 0.9582 - loss: 0.1094 - val\_accuracy: 0.9388 - val\_loss: 0.2265  
Epoch 9/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 245ms/step - accuracy: 0.9577 - loss: 0.0987 - val\_accuracy: 0.9488 - val\_loss: 0.2003  
Epoch 10/10  
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 246ms/step - accuracy: 0.9617 - loss: 0.0804 - val\_accuracy: 0.9250 - val\_loss: 0.2447

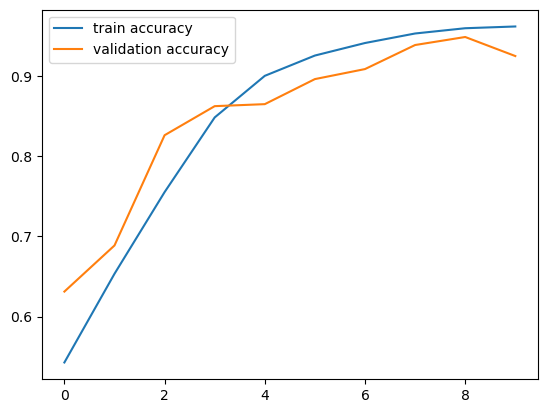
**Model Evaluation**

loss, accuracy = model.evaluate(X\_test / 255.0, Y\_test)  
print('Test Accuracy =', accuracy)

25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - accuracy: 0.9339 - loss: 0.2297  
Test Accuracy = 0.925000011920929

h = history  
  
# plot the loss value  
plt.plot(h.history['loss'], label='train loss')  
plt.plot(h.history['val\_loss'], label='validation loss')  
plt.legend()  
plt.show()  
  
# plot the accuracy value  
plt.plot(h.history['accuracy'], label='train accuracy')  
plt.plot(h.history['val\_accuracy'], label='validation accuracy')  
plt.legend()  
plt.show()





**Save the model**

model.save('model/deepfake\_detector.h5', include\_optimizer=False)

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered 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')`.

**Load the trained model**

model.compile()  
model = keras.models.load\_model('model/deepfake\_detector.h5')

WARNING:absl:No training configuration found in the save file, so the model was \*not\* compiled. Compile it manually.

**Predict function**

# Get the path of the image from user input  
def predict\_image(image\_path):  
   
 # Read the image  
 image = cv2.imread(image\_path)  
   
# # Check if the image was loaded successfully  
# if image\_ is None:  
# print("Unable to load the image. Please check the path.")  
# else:  
# # Skip displaying the image for faster processing  
  
 # Resize the image  
 image\_resized = cv2.resize(image, (128, 128))  
   
 # Scale the image  
 image\_scaled = image\_resized.astype('float32') / 255.0  
   
 # Reshape the image for the model  
 image\_reshaped = np.reshape(image\_scaled, [1, 128, 128, 3])  
   
 # Predict using the model  
 prediction = model.predict(image\_reshaped)  
   
 # Print the prediction  
 print(prediction)  
   
   
 # Get the predicted label  
 input\_pred\_label = np.argmax(prediction)  
  
 # Print the predicted label  
 print(input\_pred\_label)  
   
 # Return the predicted label if the image is real or fake  
 return "Real" if prediction[0][0] < 0.5 else "Fake"

# HTML Template  
html\_template = """  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Deepfake Detection</title>  
</head>  
<body>  
 <h1>Upload an Image for Deepfake Detection</h1>  
 <form action="/upload" method="post" enctype="multipart/form-data">  
 <input type="file" name="file" required>  
 <input type="submit" value="Upload">  
 </form>  
 {% if result %}  
 <h2>The image is: {{ result }}</h2>  
 {% endif %}  
</body>  
</html>  
"""

**Routes**

# Define the routes  
from flask import Flask, request, render\_template, render\_template\_string  
app = Flask(\_\_name\_\_)  
  
  
  
@app.route('/')  
def index():  
 return render\_template\_string(html\_template)  
  
@app.route('/upload', methods=['POST'])  
def upload\_image():  
 if 'file' not in request.files:  
 return "No file part"  
 file = request.files['file']  
 if file.filename == '':  
 return "No selected file"  
   
 filepath = os.path.join('uploads', file.filename)  
 file.save(filepath)  
   
 result = predict\_image(filepath)  
 return render\_template\_string(html\_template, result=result)  
  
# Run the app  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(port=5000, use\_reloader=False)

\* Serving Flask app '\_\_main\_\_'  
 \* Debug mode: off

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
 \* Running on http://127.0.0.1:5000  
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
 \* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
INFO:werkzeug:Press CTRL+C to quit  
127.0.0.1 - - [02/Oct/2024 02:00:17] "GET / HTTP/1.1" 200 -  
INFO:werkzeug:127.0.0.1 - - [02/Oct/2024 02:00:17] "GET / HTTP/1.1" 200 -

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step

127.0.0.1 - - [02/Oct/2024 02:00:32] "POST /upload HTTP/1.1" 200 -  
INFO:werkzeug:127.0.0.1 - - [02/Oct/2024 02:00:32] "POST /upload HTTP/1.1" 200 -

[[5.409291e-06]]  
0  
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step

127.0.0.1 - - [02/Oct/2024 02:00:41] "POST /upload HTTP/1.1" 200 -  
INFO:werkzeug:127.0.0.1 - - [02/Oct/2024 02:00:41] "POST /upload HTTP/1.1" 200 -

[[0.9766758]]  
0