```
import dlib
import cv2
import os
import re
import json
from pylab import *
from PIL import Image, ImageChops, ImageEnhance
```

```
import os
import cv2
import dlib

# Specify the base folder where you want to save the dataset
base_dataset_folder = '/content/drive/MyDrive/dataset'

train_real_folder = '/content/drive/MyDrive/output_real'
train_fake_folder = '/content/drive/MyDrive/output_fake'

# Ensure the dataset folder and subfolders exist
os.makedirs(os.path.join(base_dataset_folder, 'real'), exist_ok=True)
os.makedirs(os.path.join(base_dataset_folder, 'fake'), exist_ok=True)

real_data = [f for f in os.listdir(train_real_folder) if f.endswith('.mp4')]
fake_data = [f for f in os.listdir(train_fake_folder) if f.endswith('.mp4')]

detector = dlib.get_frontal_face_detector()

# Function to save cropped faces
def save_cropped_faces(frame, count, label, vid):
    face_rects, _, _ = detector.run(frame, 0)
    for i, d in enumerate(face_rects):
        x1 = d.left()
        y1 = d.top()
        x2 = d.right()
        y2 = d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        save_path = os.path.join(base_dataset_folder, f'{label}/{vid.split(".")[0]}_{count}.png')
        cv2.imwrite(save_path, cv2.resize(crop_img, (128, 128)))
        print(f'Saved image at: {save_path}')

# Process real videos
for vid in real_data:
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_real_folder, vid))
    frameRate = cap.get(5)
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()
        if not ret:
            break
        if frameId % ((int(frameRate) + 1) * 1) == 0:
            save_cropped_faces(frame, count, 'real', vid)
            count += 1

# Process fake videos
for vid in fake_data:
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_fake_folder, vid))
    frameRate = cap.get(5)
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()
        if not ret:
            break
        if frameId % ((int(frameRate) + 1) * 1) == 0:
            save_cropped_faces(frame, count, 'fake', vid)
            count += 1
```

```
Saved image at: /content/drive/MyDrive/dataset/real/id3_0005_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0005_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0005_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0005_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0007_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0007_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0007_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0007_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0007_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0000_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0000_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0000_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0000_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0005_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0005_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0005_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id4_0004_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id4_0004_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id4_0004_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id4_0004_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id4_0004_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0004_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0004_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0004_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0004_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0004_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0006_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0006_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0006_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0006_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0006_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id1_0005_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id1_0005_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id1_0005_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id1_0005_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id1_0005_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0009_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0009_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0009_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0009_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0009_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0001_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0001_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id2_0001_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0002_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0002_2.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0002_3.png
Saved image at: /content/drive/MyDrive/dataset/real/id3_0002_4.png
Saved image at: /content/drive/MyDrive/dataset/real/id0_0007_0.png
Saved image at: /content/drive/MyDrive/dataset/real/id0_0007_1.png
Saved image at: /content/drive/MyDrive/dataset/real/id0_0007_2.png
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
import cv2
import json
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

```
# Define input shape and data directory
input_shape = (128, 128, 3)
data_dir = '/content/drive/MyDrive/dataset'

# List real and fake data
real_data = [f for f in os.listdir(os.path.join(data_dir, 'real')) if f.endswith('.png')]
fake_data = [f for f in os.listdir(os.path.join(data_dir, 'fake')) if f.endswith('.png')]

X = []
Y = []

# Load and preprocess images
for img in real_data:
    X.append(img_to_array(load_img(os.path.join(data_dir, 'real', img), target_size=input_shape[:2])))
    Y.append(1)

for img in fake_data:
    X.append(img_to_array(load_img(os.path.join(data_dir, 'fake', img), target_size=input_shape[:2])))
    Y.append(0)

Y_val_org = Y

# Normalization
X = np.array(X) / 255.0
Y = to_categorical(Y, 2)

# Train-Test split
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=5)

# Model definition
googleNet_model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=input_shape)
googleNet_model.trainable = True

model = Sequential()
model.add(googleNet_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(units=2, activation='softmax'))

# Use the Adam optimizer with a custom learning rate
custom_optimizer = optimizers.Adam(learning_rate=1e-5, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
model.compile(loss='binary_crossentropy', optimizer=custom_optimizer, metrics=['accuracy'])

# model.summary()
```

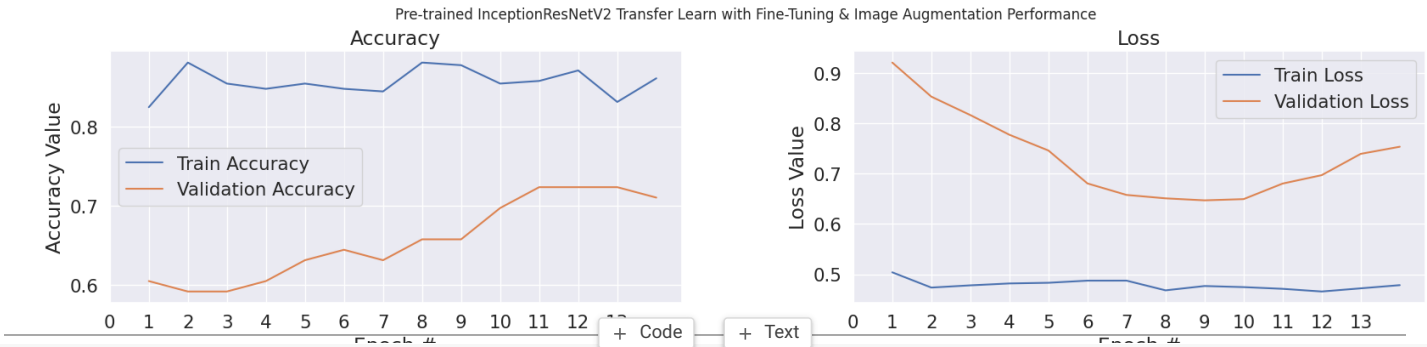
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_we
219055592/219055592 [==============================] - 1s 0us/step
```

```
# Training
early_stopping = EarlyStopping(
    monitor='val_loss',
    min_delta=0,
    patience=5,
    verbose=0,
    mode='auto'
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    min_lr=1e-6
)
```

```
EPOCHS = 20
BATCH_SIZE = 32

history = model.fit(
    X_train, Y_train,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(X_val, Y_val),
    verbose=1,
    callbacks=[early_stopping, reduce_lr]
)

# # Save the trained model
# model.save('/content/drive/MyDrive/your_model.h5')
```

```
Epoch 1/20
10/10 [==============================] - 2s 210ms/step - loss: 0.5041 - accuracy: 0.8245 - val_loss: 0.9207 - val_accuracy: 0
Epoch 2/20
10/10 [==============================] - 2s 202ms/step - loss: 0.4738 - accuracy: 0.8808 - val_loss: 0.8530 - val_accuracy: 0
Epoch 3/20
10/10 [==============================] - 2s 212ms/step - loss: 0.4783 - accuracy: 0.8543 - val_loss: 0.8163 - val_accuracy: 0
Epoch 4/20
10/10 [==============================] - 2s 236ms/step - loss: 0.4820 - accuracy: 0.8477 - val_loss: 0.7773 - val_accuracy: 0
Epoch 5/20
10/10 [==============================] - 2s 249ms/step - loss: 0.4834 - accuracy: 0.8543 - val_loss: 0.7460 - val_accuracy: 0
Epoch 6/20
10/10 [==============================] - 3s 255ms/step - loss: 0.4876 - accuracy: 0.8477 - val_loss: 0.6806 - val_accuracy: 0
Epoch 7/20
10/10 [==============================] - 2s 221ms/step - loss: 0.4876 - accuracy: 0.8444 - val_loss: 0.6578 - val_accuracy: 0
Epoch 8/20
10/10 [==============================] - 2s 202ms/step - loss: 0.4683 - accuracy: 0.8808 - val_loss: 0.6510 - val_accuracy: 0
Epoch 9/20
10/10 [==============================] - 2s 203ms/step - loss: 0.4769 - accuracy: 0.8775 - val_loss: 0.6470 - val_accuracy: 0
Epoch 10/20
10/10 [==============================] - 2s 204ms/step - loss: 0.4747 - accuracy: 0.8543 - val_loss: 0.6495 - val_accuracy: 0
Epoch 11/20
10/10 [==============================] - 2s 203ms/step - loss: 0.4714 - accuracy: 0.8576 - val_loss: 0.6804 - val_accuracy: 0
Epoch 12/20
10/10 [==============================] - 2s 218ms/step - loss: 0.4660 - accuracy: 0.8709 - val_loss: 0.6970 - val_accuracy: 0
Epoch 13/20
10/10 [==============================] - 2s 220ms/step - loss: 0.4723 - accuracy: 0.8311 - val_loss: 0.7394 - val_accuracy: 0
Epoch 14/20
10/10 [==============================] - 2s 203ms/step - loss: 0.4786 - accuracy: 0.8609 - val_loss: 0.7535 - val_accuracy: 0
```

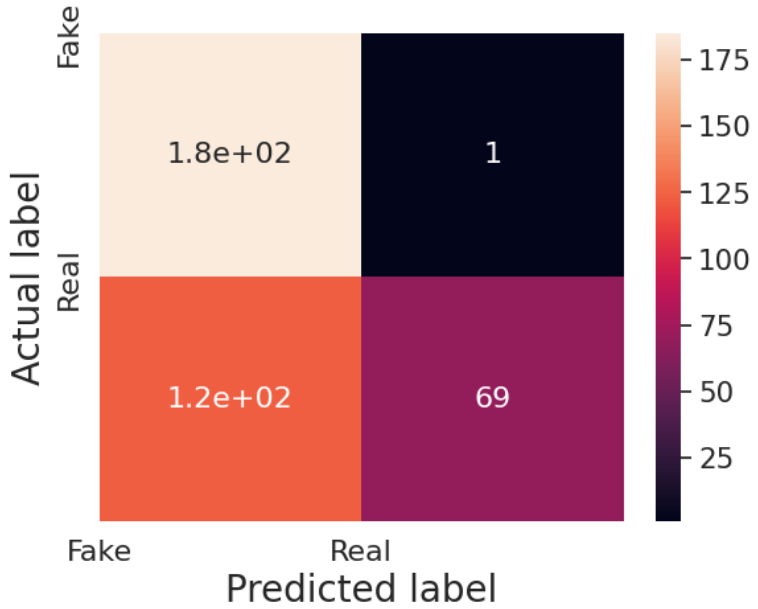
```
import matplotlib.pyplot as plt
import numpy as np


# Visualization
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 4))
t = f.suptitle('Pre-trained InceptionResNetV2 Transfer Learn with Fine-Tuning & Image Augmentation Performance', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

epoch_list = list(range(1, len(history.history['accuracy']) + 1))

ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accuracy')
ax1.set_xticks(np.arange(0, len(epoch_list), 1))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch #')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(0, len(epoch_list), 1))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch #')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")

plt.show()
```



+ Code + Text

```
# Ensure Y_val_org is a 1D array
Y_val_org = np.array(Y_val_org)

# Output confusion matrix
def print_confusion_matrix(y_true, y_pred_probs, threshold=0.5):
    y_pred_classes = (y_pred_probs[:, 1] > threshold).astype(int)

    cm = confusion_matrix(y_true, y_pred_classes)
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')

    df_cm = pd.DataFrame(cm, range(2), range(2))
    sn.set(font_scale=1.4)  # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 16})  # font size
    plt.ylabel('Actual label', size=20)
    plt.xlabel('Predicted label', size=20)
    plt.xticks(np.arange(2), ['Fake', 'Real'], size=16)
    plt.yticks(np.arange(2), ['Fake', 'Real'], size=16)
    plt.ylim([2, 0])
    plt.show()

# Use model.predict instead of model.predict_classes
print_confusion_matrix(Y_val_org, model.predict(X))
```

```
12/12 [==============================] - 1s 64ms/step
True positive =  185
False positive =  1
False negative =  123
True negative =  69
```




```
model.save('/content/drive/MyDrive/df_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 f
  saving_api.save_model(
```

```
import tensorflow as tf
import dlib
import cv2
import os
import numpy as np
from PIL import Image, ImageChops, ImageEnhance
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
```

```
tf.__version__
```

```
'2.15.0'
```

```
model = load_model('/content/drive/MyDrive/df_model.h5')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-c7161227d405> in <cell line: 1>()
----> 1 model = load_model('/content/drive/MyDrive/df_model.h5')

 2 frames 
/usr/local/lib/python3.10/dist-packages/keras/src/saving/legacy/serialization.py
in class_and_config_for_serialized_keras_object(config, module_objects,
custom_objects, printable_module_name)
    363     )
    364     if cls is None:
--> 365         raise ValueError(
    366             f"Unknown {printable_module_name}: '{class_name}'. "
    367             "Please ensure you are using a
`keras.utils.custom_object_scope` "

ValueError: Unknown layer: 'CustomScaleLayer'. Please ensure you are using a
`keras.utils.custom_object_scope` and that this object is included in the scope.
See
https://www.tensorflow.org/guide/keras/save_and_serialize#registering_the_custom_
```

```
input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/drive/MyDrive/Celeb-DF-v2/Celeb-synthesis/id0_id4_0002.mp4')
frameRate = cap.get(5)
while cap.isOpened():
    frameId = cap.get(1)
    ret, frame = cap.read()
    if ret != True:
        break
    if frameId % ((int(frameRate)+1)*1) == 0:
        face_rects, scores, idx = detector.run(frame, 0)
        for i, d in enumerate(face_rects):
            x1 = d.left()
            y1 = d.top()
            x2 = d.right()
            y2 = d.bottom()
            crop_img = frame[y1:y2, x1:x2]
            data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
            data = data.reshape(-1, 128, 128, 3)
            print(model.predict(data))
```

```
1/1 [==============================] - 1s 1s/step
[[0.5016449 0.4983551]]
1/1 [==============================] - 0s 51ms/step
[[0.70584214 0.29415783]]
1/1 [==============================] - 0s 42ms/step
[[0.68540025 0.31459975]]
1/1 [==============================] - 0s 36ms/step
[[0.715247   0.28475305]]
1/1 [==============================] - 0s 46ms/step
[[0.7827798 0.2172202]]
1/1 [==============================] - 0s 46ms/step
[[0.7768471  0.22315289]]
1/1 [==============================] - 0s 38ms/step
```

```
[[0.6587113  0.34128875]]
1/1 [==============================] - 0s 39ms/step
[[0.7132757  0.28672433]]
1/1 [==============================] - 0s 41ms/step
[[0.66198355 0.33801645]]
1/1 [==============================] - 0s 38ms/step
[[0.5809011  0.41909894]]
1/1 [==============================] - 0s 39ms/step
[[0.37282875 0.6271713 ]]
1/1 [==============================] - 0s 43ms/step
[[0.8015431  0.19845691]]
```

```
input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/drive/MyDrive/Celeb-DF-v2/YouTube-real/00001.mp4')
frameRate = cap.get(5)
while cap.isOpened():
    frameId = cap.get(1)
    ret, frame = cap.read()
    if ret != True:
        break
    if frameId % ((int(frameRate)+1)*1) == 0:
        face_rects, scores, idx = detector.run(frame, 0)
        for i, d in enumerate(face_rects):
            x1 = d.left()
            y1 = d.top()
            x2 = d.right()
            y2 = d.bottom()
            crop_img = frame[y1:y2, x1:x2]
            data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
            data = data.reshape(-1, 128, 128, 3)
            print(model.predict(data))
```

```
1/1 [==============================] - 0s 40ms/step
[[0.5580331 0.4419669]]
1/1 [==============================] - 0s 46ms/step
[[0.56275046 0.4372495 ]]
1/1 [==============================] - 0s 40ms/step
[[0.4145678  0.58543223]]
1/1 [==============================] - 0s 41ms/step
[[0.6481194  0.35188058]]
1/1 [==============================] - 0s 42ms/step
[[0.5284772 0.4715228]]
1/1 [==============================] - 0s 39ms/step
[[0.55642873 0.44357124]]
1/1 [==============================] - 0s 42ms/step
[[0.51045066 0.48954934]]
1/1 [==============================] - 0s 41ms/step
[[0.2486067 0.7513934]]
1/1 [==============================] - 0s 40ms/step
[[0.3757991 0.6242009]]
1/1 [==============================] - 0s 38ms/step
[[0.5130797 0.4869203]]
1/1 [==============================] - 0s 47ms/step
[[0.57334566 0.42665437]]
1/1 [==============================] - 0s 44ms/step
[[0.53603846 0.4639616 ]]
1/1 [==============================] - 0s 52ms/step
[[0.36650035 0.6334997 ]]
1/1 [==============================] - 0s 38ms/step
[[0.44074854 0.5592515 ]]
```