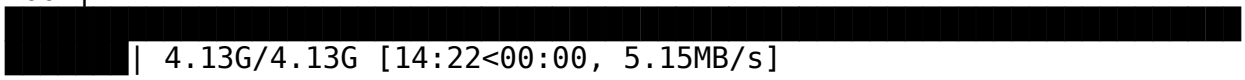
```
import numpy as np
import pandas as pd
import os
import dlib
import cv2
import os
import re
import json
from pylab import *
from PIL import Image, ImageChops, ImageEnhance

import opendatasets as od
od.download('https://www.kaggle.com/competitions/deepfake-detection-challenge/data')

Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: harshavardhangade
Your Kaggle Key: ·····
Downloading deepfake-detection-challenge.zip to .\deepfake-detection-challenge

100%|
```

```
| 4.13G/4.13G [14:22<00:00, 5.15MB/s]


Extracting archive .\deepfake-detection-challenge/deepfake-detection-challenge.zip to .\deepfake-detection-challenge

os.mkdir('/dataset1')
os.mkdir('/dataset1/real')
os.mkdir('/dataset1/fake')

train_frame_folder = './deepfake-detection-challenge/train_sample_videos'
with open(os.path.join(train_frame_folder, 'metadata.json'), 'r') as file:
    data = json.load(file)
list_of_train_data = [f for f in os.listdir(train_frame_folder) if f.endswith('.mp4')]
detector = dlib.get_frontal_face_detector()
for vid in list_of_train_data:
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_frame_folder, vid))
    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]
                if data[vid]['label'] == 'REAL':
                    cv2.imwrite('/dataset1/real/'+vid.split('.')
[0]+'_'+str(count)+'.png', cv2.resize(crop_img, (128, 128)))
                elif data[vid]['label'] == 'FAKE':
                    cv2.imwrite('/dataset1/fake/'+vid.split('.')
[0]+'_'+str(count)+'.png', cv2.resize(crop_img, (128, 128)))
                count+=1

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

input_shape = (128, 128, 3)
data_dir = '/dataset1'

real_data = [f for f in os.listdir(data_dir+'/real') if
f.endswith('.png')]
fake_data = [f for f in os.listdir(data_dir+'/fake') if
f.endswith('.png')]

X = []
Y = []

for img in real_data:
    X.append(img_to_array(load_img(data_dir+'/real/'+img)).flatten() /
255.0)
    Y.append(1)
for img in fake_data:
    X.append(img_to_array(load_img(data_dir+'/fake/'+img)).flatten() /
255.0)
    Y.append(0)
```

```
Y_val_org = Y

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

#Reshape
X = X.reshape(-1, 128, 128, 3)

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

print(Y_train)
```

```
[[1. 0.]
 [0. 1.]
 [1. 0.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

```
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import InputLayer
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping

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'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(learning_rate=1e-5,
beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 inception_resnet_v2 (Functi  (None, 2, 2, 1536)       54336736
 onal)

 global_average_pooling2d_1   (None, 1536)             0
 (GlobalAveragePooling2D)

 dense_1 (Dense)              (None, 2)                3074

=================================================================
Total params: 54,339,810
Trainable params: 54,279,266
Non-trainable params: 60,544
_________________________________________________________________
```

```
early_stopping = EarlyStopping(monitor='val_loss',
                               min_delta=0,
                               patience=2,
                               verbose=0, mode='auto')
EPOCHS = 5
BATCH_SIZE = 100
history = model.fit(X_train, Y_train, batch_size = BATCH_SIZE, epochs
= EPOCHS, validation_data = (X_val, Y_val), verbose = 1)
```

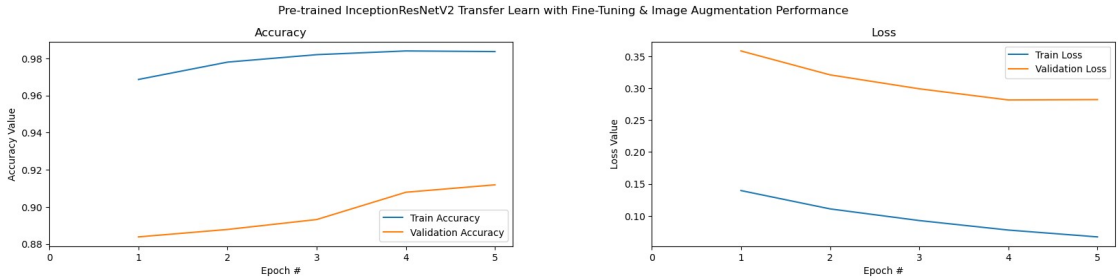
```
Epoch 1/5
30/30 [==============================] - 852s 29s/step - loss: 0.1397
- accuracy: 0.9686 - val_loss: 0.3582 - val_accuracy: 0.8838
Epoch 2/5
30/30 [==============================] - 784s 26s/step - loss: 0.1110
- accuracy: 0.9780 - val_loss: 0.3208 - val_accuracy: 0.8879
Epoch 3/5
30/30 [==============================] - 782s 26s/step - loss: 0.0927
- accuracy: 0.9820 - val_loss: 0.2990 - val_accuracy: 0.8932
Epoch 4/5
30/30 [==============================] - 784s 26s/step - loss: 0.0778
- accuracy: 0.9840 - val_loss: 0.2815 - val_accuracy: 0.9079
Epoch 5/5
30/30 [==============================] - 791s 26s/step - loss: 0.0671
- accuracy: 0.9836 - val_loss: 0.2821 - val_accuracy: 0.9119
```

```
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,EPOCHS+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, EPOCHS+1, 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, EPOCHS+1, 1))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch #')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```

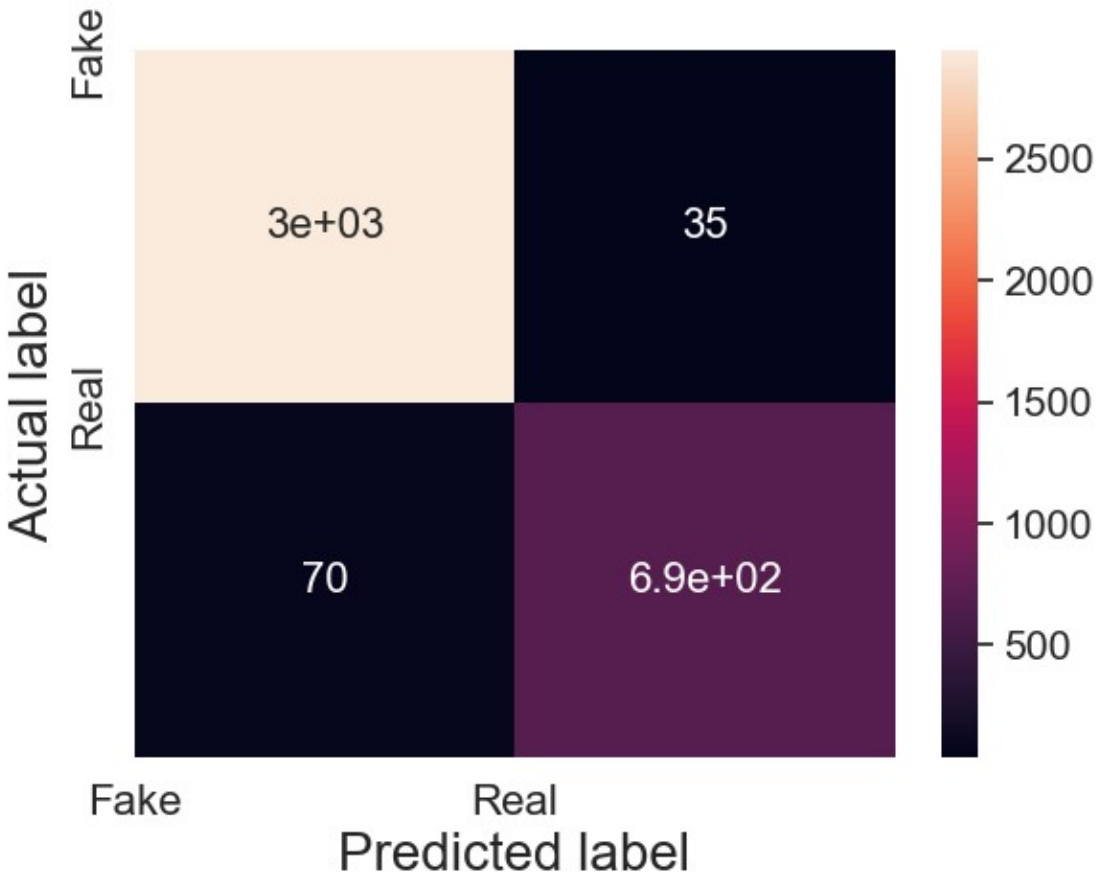



```
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')
    df_cm = pd.DataFrame(cm, range(2), 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()

classes = np.argmax(model.predict(X), axis=1)
print_confusion_matrix(Y_val_org, classes)
```

```
118/118 [==============================] - 421s 3s/step
True positive =  2951
False positive =  35
False negative =  70
```

```
True negative =  689
```




```
import sklearn.metrics
cm = sklearn.metrics.accuracy_score(Y_val_org,
np.argmax(model.predict(X), axis=1))
```

```
118/118 [==============================] - 414s 4s/step
```

```
print(cm)
```

```
0.9719626168224299
```

```
model.save('final_deepfake2-detection-model.h5')
```

```
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__

'2.9.1'

model = load_model('final_deepfake2-detection-model.h5')

input_shape = (128, 128, 3)
pr_data = []
temp = []
detector = dlib.get_frontal_face_detector()
path = input("Enter video Path for DeepFake Detection: ")#I:/Celeb-
synthesis/id3_id17_0001.mp4
cap = cv2.VideoCapture(path)
frameRate = cap.get(5)
if not cap.isOpened():
    print("Error opening video stream or file")
while True:
    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_classes(data))
            class_probabilities = model.predict(data)
            predicted_class = np.argmax(class_probabilities)
            temp.append(predicted_class)
print(temp)
predicted_classes = temp

# Count the number of times each class appears in the list
class_counts = dict()
for c in predicted_classes:
    if c in class_counts:
        class_counts[c] += 1
    else:
        class_counts[c] = 1

# Get the class that appears most often in the list
```

```python
majority_class = max(class_counts, key=class_counts.get)

# Print the appropriate message based on the majority class
if majority_class == 1:
    print("Given Video is Real")
else:
    print("Given Video is Deepfake")

# FAKE -> 0
# REAL -> 1
```

```
Enter video Path for DeepFake Detection:
I:/Celeb-synthesis/id3_id17_0001.mp4
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 251ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 234ms/step
[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0]
Given Video is Deepfake
```