Loading all the required libraries

In [107]:

```
import numpy as np
import os
import glob
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Input, Conv2D, Flatten, MaxPooling2D, Activation, Dropout, Average , BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import callbacks
from tensorflow.keras import backend as K
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import OneHotEncoder
from sklearn.utils import compute_class_weight
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score , f1_score
import scipy
from sklearn.manifold import TSNE
import seaborn as sns
from keras.utils import to_categorical
import dlib
import warnings
warnings.filterwarnings("ignore")
```

1. Final_function 1

In [108]:

```
def final_function(x):
    """
    final function 1st gives the output probablity
    on given raw input data.
    """
    image = (x)
    #loding and initializing the face detector
    detector = dlib.get_frontal_face_detector()

    # loding and initializing the pretrained facial landmark detector model from the dlib lib
    sp = dlib.shape_predictor("C:\\Users\\my pc\\Downloads\\Extra files\\shape_predictor_68_face_landmarks.dat")

    #read the image using openCV library
    img = cv2.imread(image)


    # Ask the detector to find the bounding boxes of each face. The 1 in the
    # second argument indicates that we should upsample the image 1 time. This
    # will make everything bigger and allow us to detect more faces

    det = detector(img, 1)

    # Find the 5 face landmarks we need to do the alignment.
    faces = dlib.full_object_detections()
    for detection in det:
        faces.append(sp(img, detection))
    try:
        # Get the aligned face images
        images = dlib.get_face_chips(img, faces, size=64)# wiht the size of 64 for our model.
    except RuntimeError:
        print("no face found")
        pass
    #converting detected and aligned faces into the gray scale.
    for image in images:
        a = cv2.cvtColor((image), cv2.COLOR_RGB2GRAY)
        b = a/255
        b = b.reshape(1 , 64 , 64 , 1)

    X_test = b
    #creating the model
    model = tf.keras.models.load_model('C:\\Users\\my pc\\project_first final model')
    model.load_weights("C:\\Users\\my pc\\Downloads\\CG_Face_Modified.h5")
    re = (model.predict(X_test))
    re = re.flatten()
    idx = np.argmax(re)
    b = " "
    if idx==1:
        b = "real"
    else:
        b = "Fake"
    return print("The raw image is "+b+" with the probablity of:" ,re[idx])
```

2. result_1

In [109]:

```
final_output = final_function("C:\\Users\\my pc\\Downloads\\New project_data\\Fake after pre processing\\4.jpg")
final_output
```

```
The raw image is Fake with the probablity of: 0.9992723
```

3. Final_function2

In [111]:

```
def final_function2(x , y):
    """
    This function gives the result with the final
    metric.
    """
    image = (x)
    #loding and initializing the face detector
    detector = dlib.get_frontal_face_detector()

    # loding and initializing the pretrained facial landmark detector model from the dlib lib
    sp = dlib.shape_predictor("C:\\Users\\my pc\\Downloads\\Extra files\\shape_predictor_68_face_landmarks.dat")

    #read the image using openCV library
    img = cv2.imread(image)


    # Ask the detector to find the bounding boxes of each face. The 1 in the
    # second argument indicates that we should upsample the image 1 time. This
    # will make everything bigger and allow us to detect more faces

    det = detector(img, 1)

    # Find the 5 face landmarks we need to do the alignment.
    faces = dlib.full_object_detections()
    for detection in det:
        faces.append(sp(img, detection))
    try:
        # Get the aligned face images
        images = dlib.get_face_chips(img, faces, size=64)# wiht the size of 64 for our model.
    except RuntimeError:
        print("no face found")
        pass
    #converting detected and aligned faces into the gray scale.
    for image in images:
        a = cv2.cvtColor((image), cv2.COLOR_RGB2GRAY)
        b = a/255
        b = b.reshape(1 , 64 , 64 , 1)

    X_test = b
    lst = ["real" , "fake"]
    if y in lst:
        if y=="real":
            y_true =  np.array([0 , 1])
            y_true = y_true.reshape(1,2)
        elif y =="fake":
```

```
        y_true = np.array([1 , 0])
        y_true = y_true.reshape(1,2)
    #creating the model
    model = tf.keras.models.load_model('C:\\Users\\my pc\\project_first final model')
    model.load_weights("C:\\Users\\my pc\\Downloads\\CG_Face_Modified.h5")
    result = model.evaluate(X_test , y_true)
    dt = dict(zip(model.metrics_names, result))
    return print("The accuracy of being "+y+ " is", dt["accuracy"]*100,str("%"))
else:
    print("Please enter the valid string as real or fake")
```

4. result_2

In [113]:

```
final_output2 = final_function2("C:\\Users\\my pc\\Downloads\\New project_data\\Fake after pre processing\\5.jpg" , "fake" )
final_output2
```

```
1/1 [==============================] - 0s 4ms/step - loss: 4.8758e-05 - accuracy: 1.0000
The accuracy of being fake is 100.0 %
```