**Import required libraries.**

import os

import random

from shutil import copy2

from sklearn.model\_selection import train\_test\_split

import numpy as np

from scipy.signal import convolve2d

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from PIL import Image

from sklearn.ensemble import VotingClassifier

from sklearn import svm

from sklearn.metrics import classification\_report, accuracy\_score

from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

from tensorflow.keras.applications.vgg16 import preprocess\_input

from skimage import feature, io, color

from skimage.feature import local\_binary\_pattern, hog

from tensorflow.keras.models import Model as KerasModel

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Dropout, Reshape, Concatenate, LeakyReLU

**DATA**

Extracted the images from Celeb-v2 dataset and then stored in the Hard disk.

real\_images\_base\_dir = '/Volumes/Expansion/data/extracted\_frames'

fake\_images\_base\_dir = '/Volumes/Expansion/Fake\_Extracted\_Frames'

def get\_image\_paths(base\_dir):

image\_paths = []

for root, dirs, files in os.walk(base\_dir):

for file in files:

if not file.startswith('.\_') and file.lower().endswith(('.png', '.jpg', '.jpeg')):

image\_paths.append(os.path.join(root, file))

return image\_paths

real\_image\_paths = get\_image\_paths(real\_images\_base\_dir)

fake\_image\_paths = get\_image\_paths(fake\_images\_base\_dir)

Extracted 10001 real images and 1126390 fake images from 590 real videos and 5639 fake videos respectively. since the dataset is imbalanced, the same number of random fake images are used.

def sample\_images(image\_paths, sample\_size):

if len(image\_paths) > sample\_size:

return random.sample(image\_paths, sample\_size)

return image\_paths

min\_image\_count = min(len(real\_image\_paths), len(fake\_image\_paths))

real\_images\_sample = sample\_images(real\_image\_paths, min\_image\_count)

fake\_images\_sample = sample\_images(fake\_image\_paths, min\_image\_count)

print(len(real\_images\_sample))

print(len(fake\_images\_sample))

10001

10001

After under sampling, we got an equal number of samples from both the real and fake classes, totalling 10001 real images and 10001 fake images. The size of all extracted image is 518x498.

**DATA SPLIT INTO TRAIN and TEST**

Split the data into training and testing sets with an 80:20 ratio.

# Split real images

real\_images\_train, real\_images\_test = train\_test\_split(real\_images\_sample, test\_size=0.2, random\_state=42)

# Split fake images

fake\_images\_train, fake\_images\_test = train\_test\_split(fake\_images\_sample, test\_size=0.2, random\_state=42)

Number of real images in training set 8000

Number of real images in testing set 2001

Number of fake images in training set 8000

Number of fake images in testing set 2001

After splitting, the training set contains 8000 real images and 8000 fake images, while the testing set consists of 2001 real images and 2001 fake images.

**MesoNet**

Model architecture used from the github repository [MesoNet](https://github.com/DariusAf/MesoNet/tree/master).I added 'base\_model' and 'feature\_model' in the Meso model. While base\_model represents the full classification model, feature\_model outputs feature from the last convolutional layer (x4).

IMGWIDTH = 256

class Classifier:

def \_\_init\_\_(self):

self.model = None

def predict(self, x):

return self.model.predict(x)

def fit(self, x, y, batch\_size=32, epochs=10):

return self.model.fit(x, y, batch\_size=batch\_size, epochs=epochs)

def get\_accuracy(self, x, y):

return self.model.evaluate(x, y)

def load(self, path):

self.model.load\_weights(path)

class Meso4(Classifier):

def \_\_init\_\_(self, learning\_rate=0.001):

super().\_\_init\_\_()

self.base\_model, self.feature\_model = self.init\_model()

optimizer = Adam(learning\_rate=learning\_rate)

self.base\_model.compile(optimizer=optimizer, loss='mean\_squared\_error', metrics=['accuracy'])

self.model = self.base\_model # Setting the main model to the full base model

def init\_model(self):

x = Input(shape=(IMGWIDTH, IMGWIDTH, 3))

x1 = Conv2D(8, (3, 3), padding='same', activation='relu')(x)

x1 = BatchNormalization()(x1)

x1 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x1)

x2 = Conv2D(8, (5, 5), padding='same', activation='relu')(x1)

x2 = BatchNormalization()(x2)

x2 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x2)

x3 = Conv2D(16, (5, 5), padding='same', activation='relu')(x2)

x3 = BatchNormalization()(x3)

x3 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x3)

x4 = Conv2D(16, (5, 5), padding='same', activation='relu')(x3)

x4 = BatchNormalization()(x4)

x4 = MaxPooling2D(pool\_size=(4, 4), padding='same')(x4)

feature\_output = x4

y = Flatten()(x4)

y = Dropout(0.5)(y)

y = Dense(16)(y)

y = LeakyReLU(alpha=0.1)(y)

y = Dropout(0.5)(y)

y = Dense(1, activation='sigmoid')(y)

base\_model = KerasModel(inputs=x, outputs=y)

feature\_model = KerasModel(inputs=x, outputs=feature\_output)

return base\_model, feature\_model

meso\_model = Meso4()

**Training Phase**

To pre-process real and fake images for training the MesoNet model, the 'preprocess\_images' function is utilized. This function first resizes the images to 256x256, then converts them into arrays, and finally normalizes the pixel values to between 0 and 1. All the processed images are added to a list and returned as a stack of pre-processed images. Subsequently, these processed 16000 train\_images, along with their corresponding labels (0 for fake and 1 for real), are fed into the MesoNet network for training.

def preprocess\_images(image\_paths, target\_size=(256, 256)):

images = []

for img\_path in image\_paths:

img = load\_img(img\_path, target\_size=target\_size)

img = img\_to\_array(img)

img /= 255.0

images.append(img)

return np.stack(images)

**Training MesoNet Model**

train\_images = preprocess\_images(real\_images\_train + fake\_images\_train)

y\_train=np.concatenate([np.ones(len(real\_images\_train)),np.zeros(len(fake\_images\_train))])

For training purposes, the real samples are labelled as 1 and the fake samples are labelled as 0.

meso\_model.base\_model.fit(train\_images, y\_train, epochs=10, batch\_size=32)

MesoNet base model is trained on the training images (train\_images) and their corresponding labels (y\_train) for 10 epochs with a batch size of 32.

**Local Binary Patterns (LBP)**

The 'compute\_lbp\_features' function calculates the Local Binary Pattern (LBP) for each image and generates a histogram with 1024 bins, resulting in 1024 features for each image. Before computing the LBP, the image undergoes pre-processing using the 'load\_image' function, which converts it to grayscale, resizes it to 256x256, and converts it to an array.

(Ref : [LBP](https://scikit-image.org/docs/stable/api/skimage.feature.html#skimage.feature.local_binary_pattern) , [implementation of LBP](https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/) )

def load\_image(image\_path, target\_size=(256, 256)):

with Image.open(image\_path) as img:

img = img.convert('L')

img = img.resize(target\_size)

return np.array(img)

def compute\_lbp\_features(images, radius=3, points=24, method='uniform', num\_bins=1024):

"""Compute LBP features for a list of images."""

lbp\_features = []

for img\_path in images:

img = load\_image(img\_path)

lbp = local\_binary\_pattern(img, P=points, R=radius, method=method)

hist, \_ = np.histogram(lbp.ravel(), bins=num\_bins, range=(0, num\_bins), density=True)

lbp\_features.append(hist)

return lbp\_features

**Feature Extraction of train data Using LBP**

real\_train\_lbp = compute\_lbp\_features(real\_images\_train)

Here, the feature vectors for 8000 real images each consist of 1024 features.

fake\_train\_lbp = compute\_lbp\_features(fake\_images\_train)

Here, the feature vectors for 8000 fake images each consist of 1024 features.

def load\_and\_preprocess\_image(image\_path, target\_size=(256, 256)):

img = load\_img(image\_path, target\_size=target\_size)

img\_array = img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0)

img\_array /= 255.0

return img\_array

**Feature Extraction of train data using MesoNet**

The 'extract\_features\_with\_mesonet' function is used to extract features from each image using the MesoNet network. Before feeding the image into MesoNet, it undergoes pre-processing using the 'load\_and\_preprocess\_image' function. This function resizes it to the 256x256 size, converts it to an array, and normalizes the pixel values to between 0 and 1. Subsequently, the pre-processed image is input into MesoNet to extract features from its hidden layer(x4). Finally, the extracted feature vector is flattened into a 1D array.

def extract\_features\_with\_mesonet(feature\_extractor, image\_paths):

features = []

for img\_path in image\_paths:

if not os.path.basename(img\_path).startswith('.\_'):

try:

img\_array = load\_and\_preprocess\_image(img\_path)

image\_features = feature\_extractor.feature\_model.predict(img\_array)

features.append(image\_features.flatten())

except Exception as e:

print(f"Error processing image {img\_path}: {e}")

continue

return np.stack(features)

meso\_trainfeatures = extract\_features\_with\_mesonet(meso\_model,real\_images\_train)

meso\_trainfeatures.shape

(8000, 1024)

Here, we extracted 1024 features from MesoNet for each image in 8000 real sample.

meso\_fake\_trainfeatures =extract\_features\_with\_mesonet(meso\_model,fake\_images\_train)

Here, we extracted 1024 features from MesoNet for each image in 8000 real sample.

**Combining Features from LBP and MesoNet**

real\_train = np.concatenate((real\_train\_lbp, meso\_trainfeatures), axis=1)

Here, the features extracted from both the LBP and the MesoNet network are combined for both real and fake samples.

X\_train = np.vstack((real\_train, fake\_train)

This function combines the real and fake training images into a single array along their vertical axis. it consists of 16,000 samples with 2048 features each.

**Training SVM Model**

clf = svm.SVC(kernel='linear')

clf.fit(X\_train, y\_train)

SVM classifier with a linear kernel used and trained the classifier on the training data (X\_train and y\_train) using the fit method.

**Testing Phase**

**Feature extraction of testing data using LBP**

real\_test\_lbp = compute\_lbp\_features(real\_images\_test)

The 1024 features for real samples have been extracted.

fake\_test\_lbp = compute\_lbp\_features(fake\_images\_test)

The 1024 features for fake samples have been extracted.

**Feature extraction of testing data using MesoNet**

meso\_real\_testfeatures = extract\_features\_with\_mesonet(meso\_model,real\_images\_test)

Extracted 1024 features for each real image in total of 2001.

meso\_fake\_testfeatures = extract\_features\_with\_mesonet(meso\_model,fake\_images\_test)

Extracted 1024 features for each fake image in total of 2001.

**Combining Features from LBP and MesoNet**

real\_test = np.concatenate((real\_test\_lbp, meso\_real\_testfeatures), axis=1)

Here, the features extracted from both the LBP and the MesoNet network are combined for both real and fake testing samples.

X\_test = np.vstack((real\_test, fake\_test))

This function combines the real and fake testing images into a single array along their vertical axis. it consists of 2001 samples with 2048 features each.

y\_test = np.concatenate([np.ones(len(real\_test)), np.zeros(len(fake\_test))])

the real samples are labelled as 1, and the fake samples are labelled as 0.

**Predicting using SVM Model**

y\_pred\_test = clf.predict(X\_test)

Used the clf (trained classifier) to predict the labels on the test data (X\_test) and stored the predictions in the variable 'y\_pred\_test'

print("Test Set Classification Report:\n", classification\_report(y\_test, y\_pred\_test)

Test Set Classification Report:

precision recall f1-score support

0.0 1.00 0.99 0.99 2001

1.0 0.99 1.00 0.99 2001

accuracy 0.99 4002

macro avg 0.99 0.99 0.99 4002

weighted avg 0.99 0.99 0.99 4002

print("Test Set Accuracy:", accuracy\_score(y\_test, y\_pred\_test))

Test Set Accuracy: 0.9932533733133433

The accuracy of the classifier on the test set is 98.93%.