```
##############
# DEEPFAKE 1 #
##############

# This code is optimised and changed in accordance with the project requirements, code changes are mentioned in Bitbucket repo
# IMPORTANT: the code requires the use of radialProfile.py, it is present on the speed node under user "Mukul Prabhu", for usi
# The algorithm tests 2 pickle files from 2 datasets and uses images from the data sets itself and other datasets too, details
# IMPORTANT: Pickle files are hardcoded in the notebook for better usability, please change the paths accordingly
# Pickle files are present in bitbucket and on the speed node
# IMPORTANT: To use on speed node change the kernel to Mukul's Kernel and activate mukulenv to avoid dependency issues

##############
# DEEPFAKE 1 #
##############
```

```
################################################
# Calculations for test image, used for graph #
################################################

def preprocess_image(image_path):
    epsilon = 1e-8
    N = 80
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    fshift += epsilon

    magnitude_spectrum = 20 * np.log(np.abs(fshift))
    psd1D = radialProfile.azimuthalAverage(magnitude_spectrum)

    # Calculate the azimuthally averaged 1D power spectrum
    points = np.linspace(0, N, num=psd1D.size)
    xi = np.linspace(0, N, num=N)

    interpolated = griddata(points, psd1D, xi, method='cubic')
    interpolated /= interpolated[0]

    return interpolated, psd1D
```

```
####################################
# Calcultions for dataset spectrum #
####################################

def dataset_spectrum_calculation(the_pickel_file):

  pkl_file = open(the_pickel_file, 'rb')
  data = pickle.load(pkl_file)
  pkl_file.close()
  X = data["data"]
  y = data["label"]

  num = int(X.shape[0]/2)
  num_feat = X.shape[1]

  psd1D_org_0 = np.zeros((num,num_feat))
  psd1D_org_1 = np.zeros((num,num_feat))
  psd1D_org_0_mean = np.zeros(num_feat)
  psd1D_org_0_std = np.zeros(num_feat)
  psd1D_org_1_mean = np.zeros(num_feat)
  psd1D_org_1_std = np.zeros(num_feat)

  cont_0=0
  cont_1=0

  # We separate real and fake using the label
  for x in range(X.shape[0]):
      if y[x]==0:
          psd1D_org_0[cont_0,:] = X[x,:]
          cont_0+=1
      elif y[x]==1:
          psd1D_org_1[cont_1,:] = X[x,:]
          cont_1+=1

  # We compute statistcis
  for x in range(num_feat):
      psd1D_org_0_mean[x] = np.mean(psd1D_org_0[:,x])
      psd1D_org_0_std[x]= np.std(psd1D_org_0[:,x])
      psd1D_org_1_mean[x] = np.mean(psd1D_org_1[:,x])
```

```
    psd1D_org_1_std[x]= np.std(psd1D_org_1[:,x])

return psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat
```

```
###############
# Classifiers #
###############

def train_test(the_pickel_file, preprocessed_image, psd1D):

  # Reshape the image to match the model input
  preprocessed_image = preprocessed_image.reshape(1, -1)

  # Load the trained model (assuming it has been saved as a pickle file)
  num = 10
  LR = 0 # Logistic Regression
  SVM = 0 # Support Vector Machine with Linear Kernel
  SVM_r = 0 # Support Vector Machine with Radial Basis Function
  SVM_p = 0 # Support Vector Machine with Polnomial Kernel


  for z in range(num):
      # read python dict back from the file
      pkl_file = open(the_pickel_file, 'rb')

      data = pickle.load(pkl_file)

      pkl_file.close()
      X = data["data"]
      y = data["label"]


      try:

          from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

          from sklearn.svm import SVC
          svclassifier = SVC(kernel='linear')
          svclassifier.fit(X_train, y_train)
          #Uncomment the next line to get test results
          #print('Accuracy on test set: {:.3f}'.format(svclassifier.score(X_test, y_test)))

          from sklearn.svm import SVC
          svclassifier_r = SVC(C=6.37, kernel='rbf', gamma=0.86)
          svclassifier_r.fit(X_train, y_train)
          #Uncomment the next line to get test results
          #print('Accuracy on test set: {:.3f}'.format(svclassifier_r.score(X_test, y_test)))

          from sklearn.svm import SVC
          svclassifier_p = SVC(kernel='poly')
          svclassifier_p.fit(X_train, y_train)
          #Uncomment the next line to get test results
          #print('Accuracy on test set: {:.3f}'.format(svclassifier_p.score(X_test, y_test)))

          from sklearn.linear_model import LogisticRegression
          logreg = LogisticRegression(solver='liblinear', max_iter=1000)
          logreg.fit(X_train, y_train)
          #Uncomment the next line to get test results
          #print('Accuracy on test set: {:.3f}'.format(logreg.score(X_test, y_test)))


          SVM+=svclassifier.score(X_test, y_test)
          SVM_r+=svclassifier_r.score(X_test, y_test)
          SVM_p+=svclassifier_p.score(X_test, y_test)
          LR+=logreg.score(X_test, y_test)


      except:
          num-=1
          print(num)

  try:

    # Make predictions with the trained models
    prediction_svm = svclassifier.predict(preprocessed_image)
    prediction_svm_r = svclassifier_r.predict(preprocessed_image)
    prediction_svm_p = svclassifier_p.predict(preprocessed_image)
    prediction_lr = logreg.predict(preprocessed_image)

    # Uncomment the next lines to get prediction results
    # print("Prediction for the image (Linear SVM):", prediction_svm)
    # print("Prediction for the image (RBF SVM):", prediction_svm_r)
    # print("Prediction for the image (Polynomial SVM):", prediction_svm_p)
    # print("Prediction for the image (Logistic Regression)", prediction_lr)
```

```
  except:
      print("")
```

```
#########################
# Plot Dataset spectrum #
#########################

def dataset_spectrum_graph(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, the_pickel_file):
  x = np.arange(0, num_feat, 1)
  fig, ax = plt.subplots(figsize=(15, 9))
  ax.plot(x, psd1D_org_0_mean, alpha=0.5, color='red', label='Fake', linewidth =2.0)
  ax.fill_between(x, psd1D_org_0_mean - psd1D_org_0_std, psd1D_org_0_mean + psd1D_org_0_std, color='red', alpha=0.2)
  ax.plot(x, psd1D_org_1_mean, alpha=0.5, color='blue', label='Real', linewidth =2.0)
  ax.fill_between(x, psd1D_org_1_mean - psd1D_org_1_std, psd1D_org_1_mean + psd1D_org_1_std, color='blue', alpha=0.2)
  ax.legend(loc='best', prop={'size': 20})
  plt.tick_params(axis='x', labelsize=20)
  plt.tick_params(axis='y', labelsize=20)
  ax.legend(loc='best', prop={'size': 20})
  plt.xlabel("Spatial Frequency", fontsize=20)
  plt.ylabel("Power Spectrum", fontsize=20)
  plt.title("Power Spectrum of the Dataset: " + the_pickel_file, fontsize=20)
  plt.show()
```

```
############################
# Plot Test Image Spectrum #
############################

def test_image_graph(psd1D, image_path, the_pickel_file):
  x = np.arange(0, psd1D.shape[0], 1)
  plt.figure(figsize=(15, 9))
  plt.plot(x, psd1D, alpha=0.5, color='green', label=image_path, linewidth=2.0)
  plt.tick_params(axis='x', labelsize=20)
  plt.tick_params(axis='y', labelsize=20)
  plt.legend(loc='best', prop={'size': 20})
  plt.xlabel("Spatial Frequency", fontsize=20)
  plt.ylabel("Power Spectrum", fontsize=20)
  plt.title("Power Spectrum of the Dataset: " + the_pickel_file, fontsize=20)
  plt.show()
```

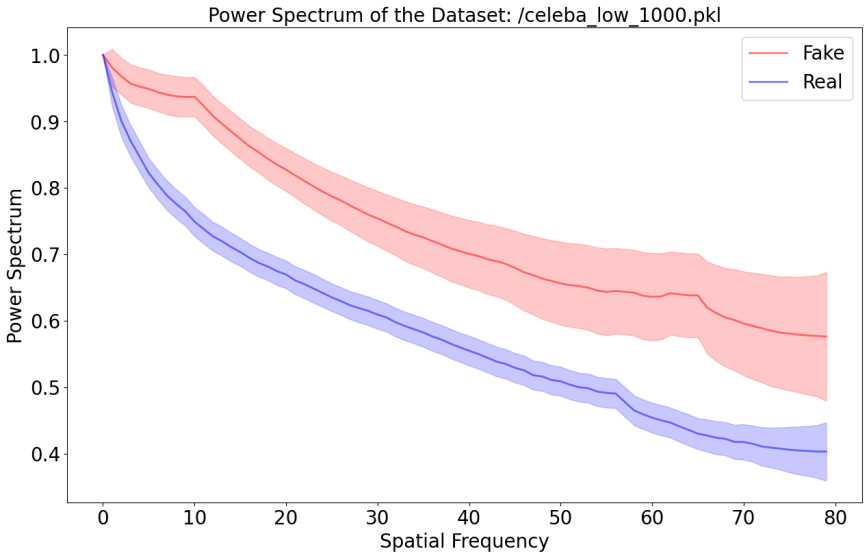
```
########################
# Combining the results #
########################

def plot_combination_graph(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, psd1D, image_path
  x = np.arange(0, num_feat, 1)
  fig, ax = plt.subplots(figsize=(15, 9))
  ax.plot(x, psd1D_org_0_mean, alpha=0.5, color='red', label='Fake', linewidth =2.0)
  ax.fill_between(x, psd1D_org_0_mean - psd1D_org_0_std, psd1D_org_0_mean + psd1D_org_0_std, color='red', alpha=0.2)
  ax.plot(x, psd1D_org_1_mean, alpha=0.5, color='blue', label='Real', linewidth =2.0)
  ax.fill_between(x, psd1D_org_1_mean - psd1D_org_1_std, psd1D_org_1_mean + psd1D_org_1_std, color='blue', alpha=0.2)
  x = np.arange(0, psd1D.shape[0], 1)
  ax.plot(x, psd1D, alpha=0.5, color='green', label=image_path, linewidth=2.0)
  plt.tick_params(axis='x', labelsize=20)
  plt.tick_params(axis='y', labelsize=20)
  ax.legend(loc='best', prop={'size': 20})
  plt.title("Power Spectrum of the Dataset: " + the_pickel_file, fontsize=20)
  plt.xlabel("Spatial Frequency", fontsize=20)
  plt.ylabel("Power Spectrum", fontsize=20)
  plt.show()
```

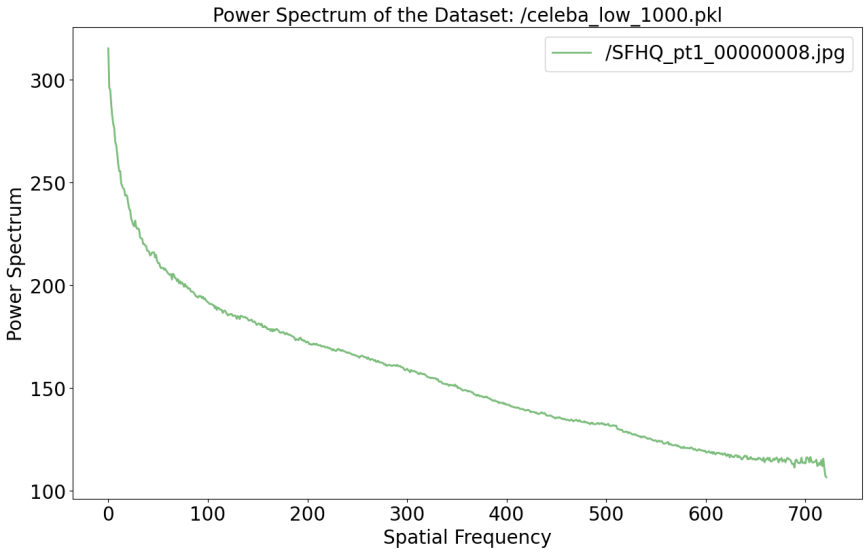
```
###################################
# Function to plot all the graphs #
###################################

def plot_all_graphs(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, psd1D, image_path, the_p
  dataset_spectrum_graph(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, the_pickel_file)
  test_image_graph(psd1D, image_path, the_pickel_file)
  plot_combination_graph(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, psd1D, image_path,
```

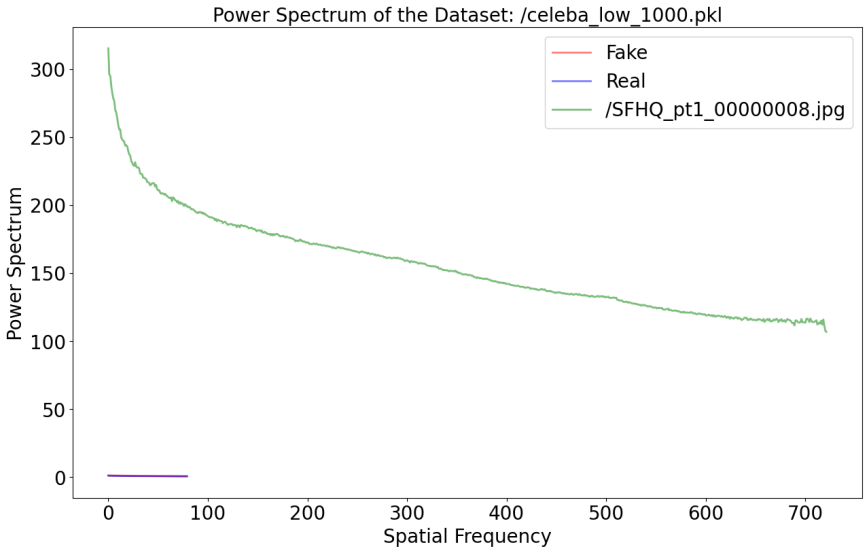
```
#############
# MAIN CELL #
#############

import cv2
import numpy as np
import radialProfile
from scipy.interpolate import griddata
```

```
import pickle
import matplotlib.pyplot as plt

# Hardcoded pre-tarined pickle files, change the path as required
list_of_pickles = ['/celeba_low_1000.pkl', '/dataset_freq_1000.pkl']

while (True):

  try:
    image_path = input("Enter the file path of the image to be tested\nOr type 'exit' to quit\n")


    if image_path == 'exit':
      break

    preprocessed_image, psd1D = preprocess_image(image_path)

    for the_pickel_file in list_of_pickles:
      print("Results for the dataset: " + the_pickel_file + " and algorithm: Unmasking Deepfakes with simple features are\n")

      psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat = dataset_spectrum_calculation(the_pickel
      train_test(the_pickel_file, preprocessed_image, psd1D)
      plot_all_graphs(psd1D_org_0_mean, psd1D_org_0_std, psd1D_org_1_mean, psd1D_org_1_std, num_feat, psd1D, image_path, the_p
      print("\n")
  except:
    print("")
```

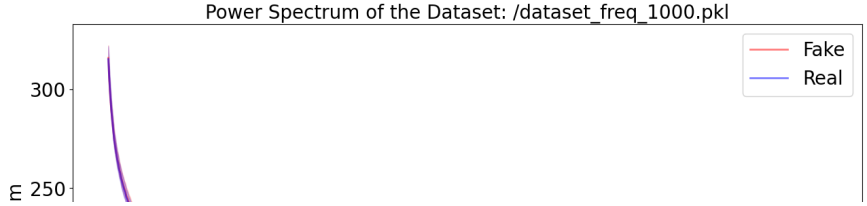
```
Enter the file path of the image to be tested
Or type 'exit' to quit/SFHQ_pt1_00000008.jpg
Results for the dataset: /celeba_low_1000.pkl and algorithm: Unmasking Deepfak
```

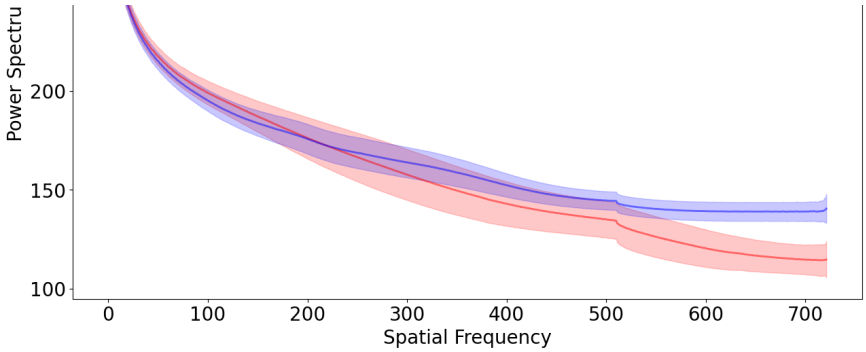


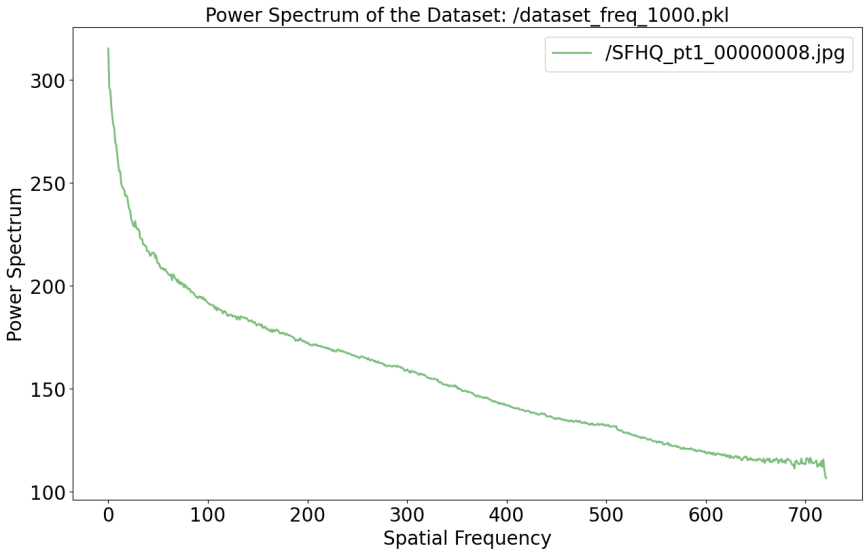


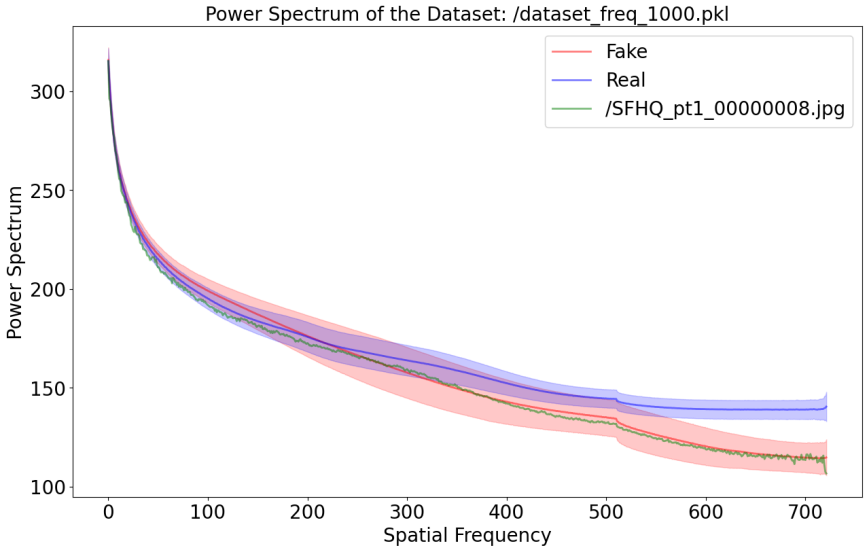



```
Results for the dataset: /dataset_freq_1000.pkl and algorithm: Unmasking Deepf
```










```
Enter the file path of the image to be tested
Or type 'exit' to quitexit
```