Deep Texture Features for Robust Face Spoofing Detection

Now-a-days biometric systems are useful in recognizing person’s identity but criminals change their appearance in behaviour and psychological to deceive recognition system. To overcome from this problem this paper introduce new technique called Deep Texture Features extraction from images and then building train model using CNN (Convolution Neural Networks) algorithm. This technique refer as LBPNet or NLBPNet as this technique heavily dependent on features extraction using LBP (Local Binary Pattern) algorithm.

In this project we are designing LBP Based Convolution Neural Network called LBPNET to detect face spoofing. Here first we will extract LBP from images and then train LBP descriptor images with Convolution Neural Network to generate training model. Whenever we upload new test image then that test image will be applied on training model to detect whether test image contains spoof image or non-spoof image. Below we can see some details on LBP.

Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision and is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

The LBP feature vector, in its simplest form, is created in the following manner:

Divide the examined window into cells (e.g. 16x16 pixels for each cell).

For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.

Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).

Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.

Optionally normalize the histogram.

Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

The feature vector can now be processed using the Support vector machine, extreme learning machines, or some other machine learning algorithm to classify images. Such classifiers can be used for face recognition or texture analysis.

A useful extension to the original operator is the so-called uniform pattern,[8] which can be used to reduce the length of the feature vector and implement a simple rotation invariant descriptor. This idea is motivated by the fact that some binary patterns occur more commonly in texture images than others. A local binary pattern is called uniform if the binary pattern contains at most two 0-1 or 1-0 transitions. For example, 00010000 (2 transitions) is a uniform pattern, but 01010100 (6 transitions) is not. In the computation of the LBP histogram, the histogram has a separate bin for every uniform pattern, and all non-uniform patterns are assigned to a single bin. Using uniform patterns, the length of the feature vector for a single cell reduces from 256 to 59. The 58 uniform binary patterns correspond to the integers 0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254 and 255.

CNN working procedure

To demonstrate how to build a convolutional neural network based image classifier, we shall build a 6 layer neural network that will identify and separate one image from other. This network that we shall build is a very small network that we can run on a CPU as well. Traditional neural networks that are very good at doing image classification have many more parameters and take a lot of time if trained on normal CPU. However, our objective is to show how to build a real-world convolutional neural network using TENSORFLOW.

Neural Networks are essentially mathematical models to solve an optimization problem. They are made of neurons, the basic computation unit of neural networks. A neuron takes an input (say x), do some computation on it (say: multiply it with a variable w and adds another variable b) to produce a value (say; z= wx + b). This value is passed to a non-linear function called activation function (f) to produce the final output (activation) of a neuron. There are many kinds of activation functions. One of the popular activation function is Sigmoid. The neuron which uses sigmoid function as an activation function will be called sigmoid neuron. Depending on the activation functions, neurons are named and there are many kinds of them like RELU, TanH.

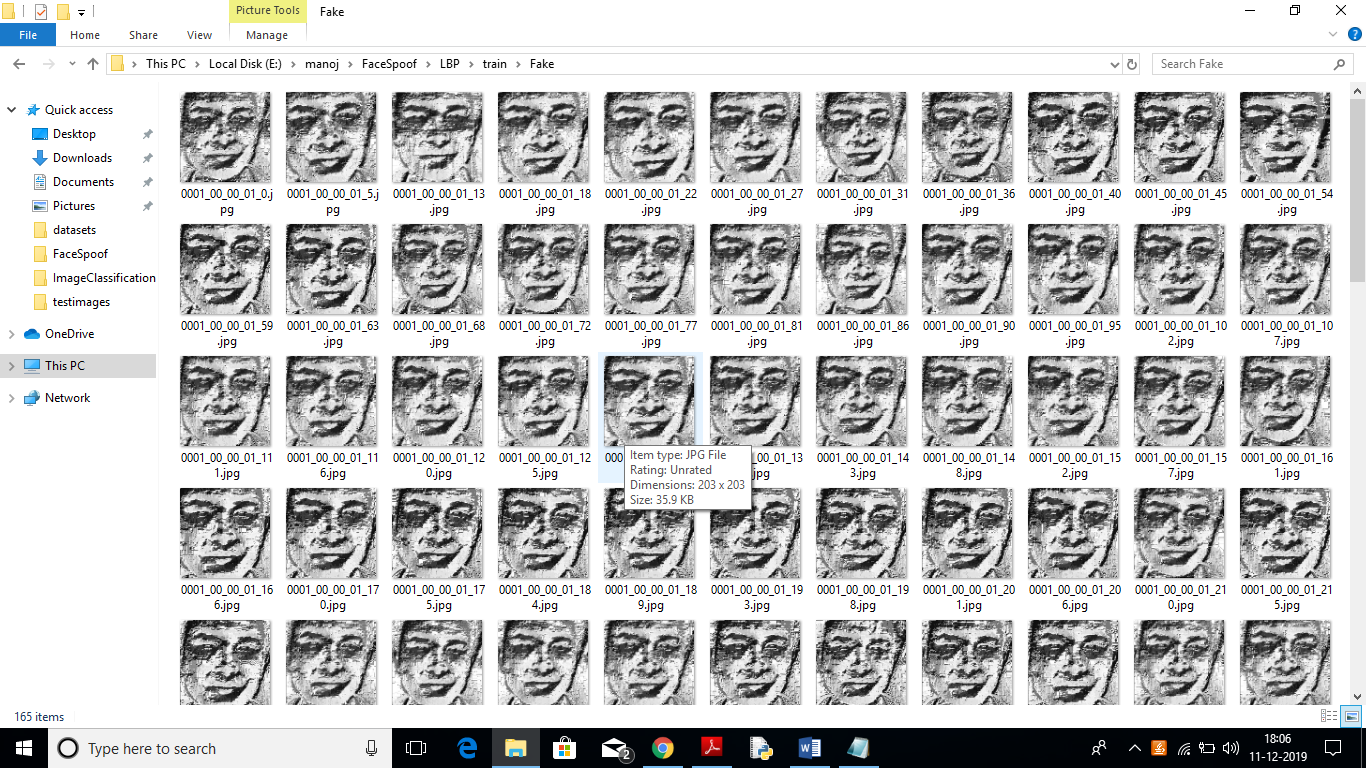
If you stack neurons in a single line, it’s called a layer; which is the next building block of neural networks. See below image with layers



To predict image class multiple layers operate on each other to get best match layer and this process continues till no more improvement left.

Dataset Details:

In this paper author has used NUAA Photograph Imposter (fake) Database with images obtained from real and fake faces. We also used images and convert that image into LBP format. Below are some images from LBP folder



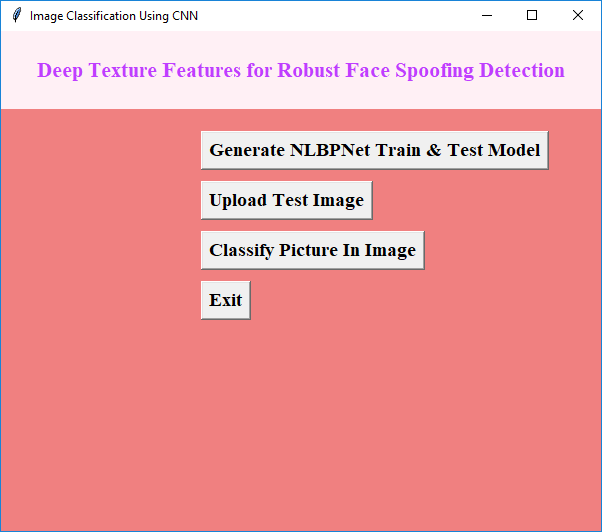
All this fake and real images you can see inside ‘LBP/train’ folder.

This project consists of following modules:

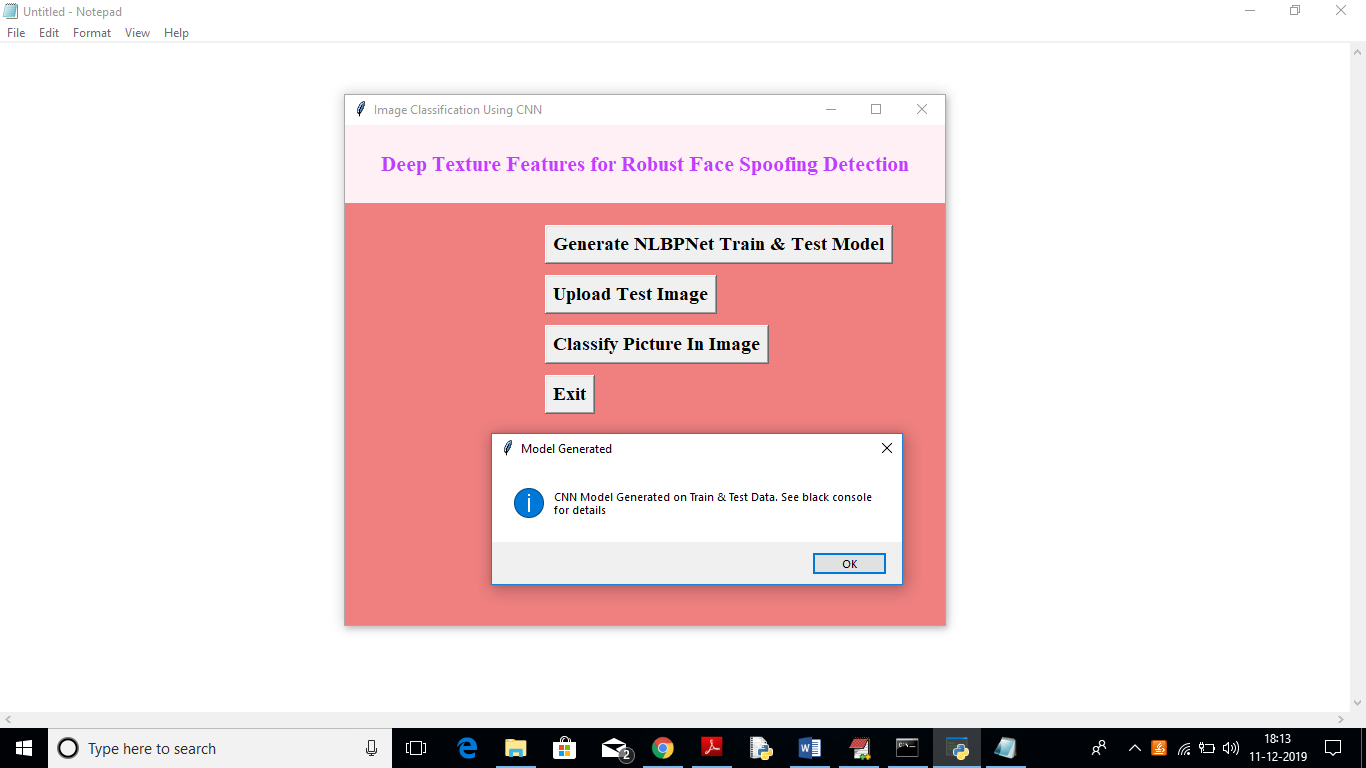
1. Generate NLBPNet Train & Test Model: in this module we will read all LBP images from LBP folder and then train CNN model with all those images.
2. Upload Test Image: In this module we will upload test image from ‘testimages’ folder. Application will read this image and then extract Deep Textures Features from this image using LBP algorithm.
3. Classify Picture In Image: This module apply test image on CNN train model to predict whether test image contains spoof or non-spoof face.

Screen shots

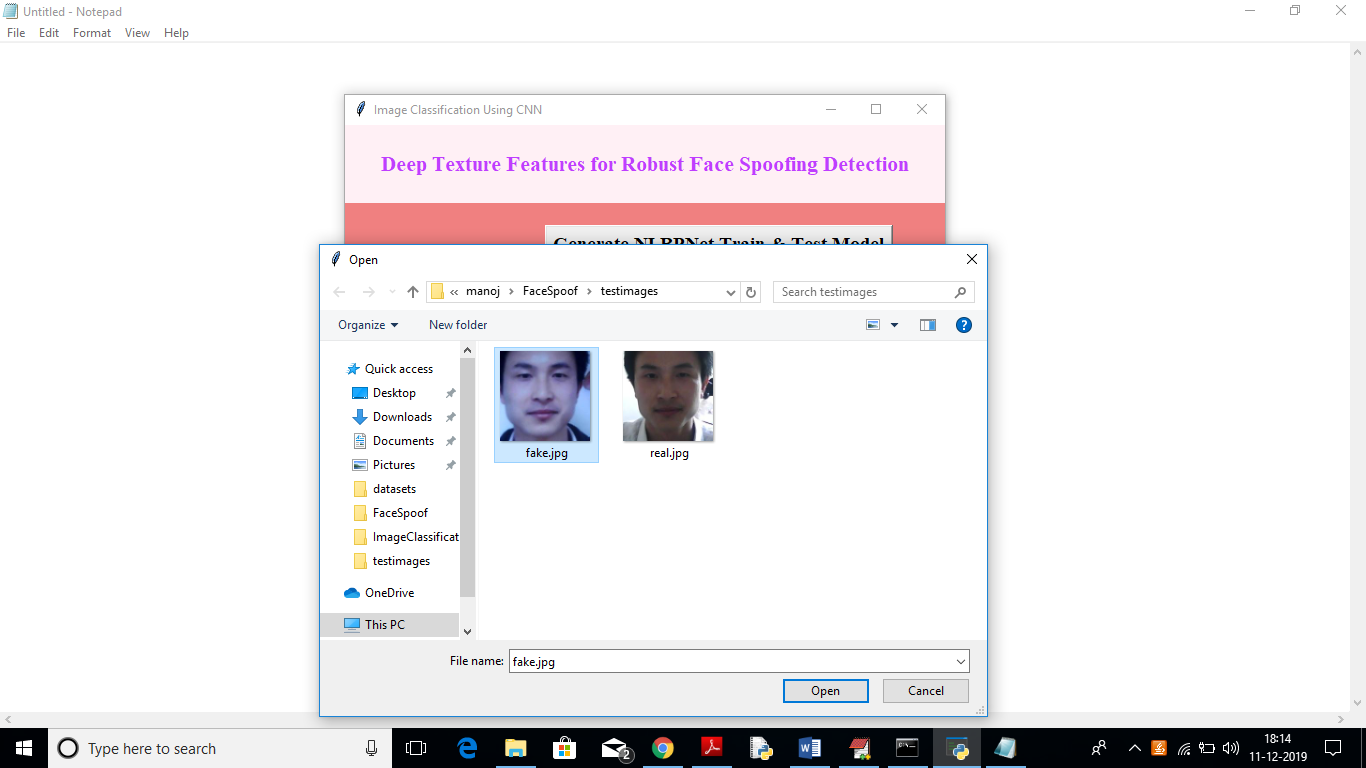
To run this project double click on ‘run.bat’ file to get below screen



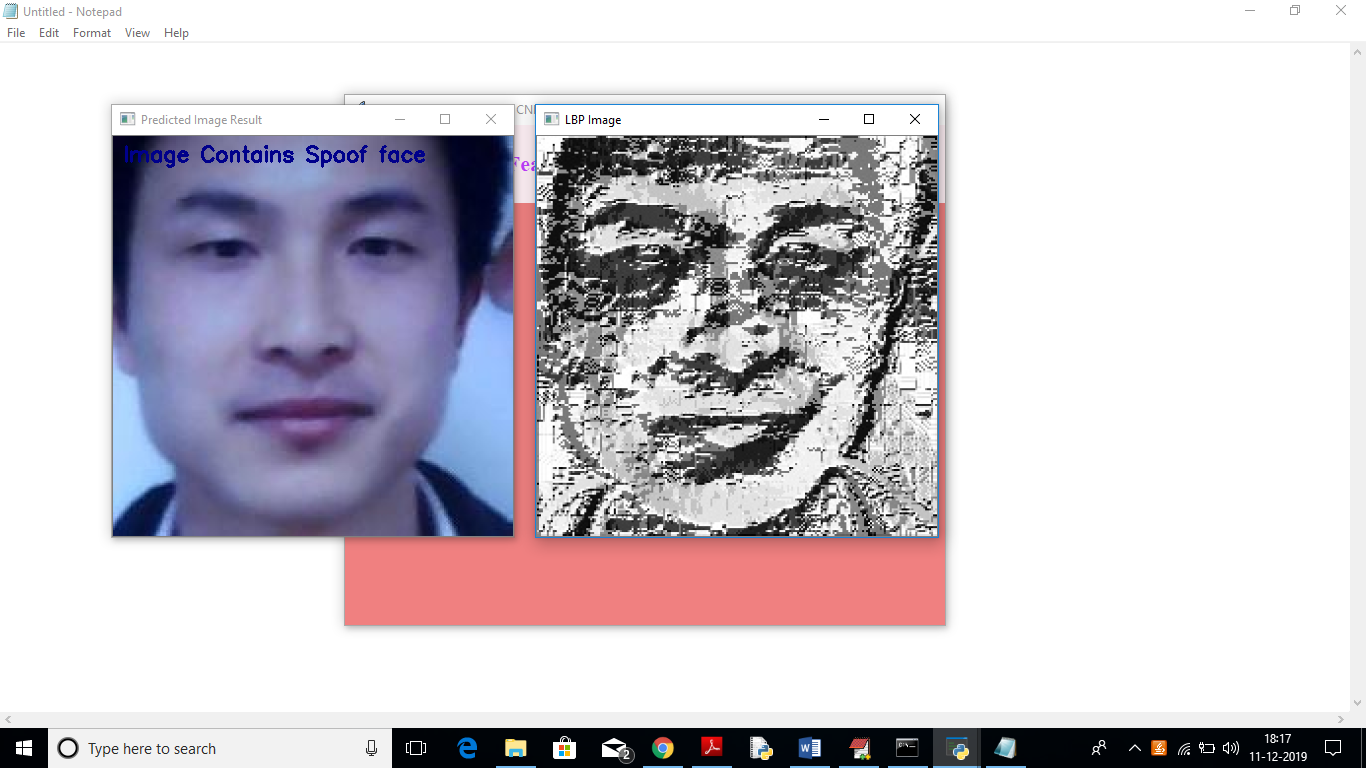
In above screen click on ‘Generate NLBPNet Train & Test Model’ button to generate CNN model using LBP images contains inside LBP folder.



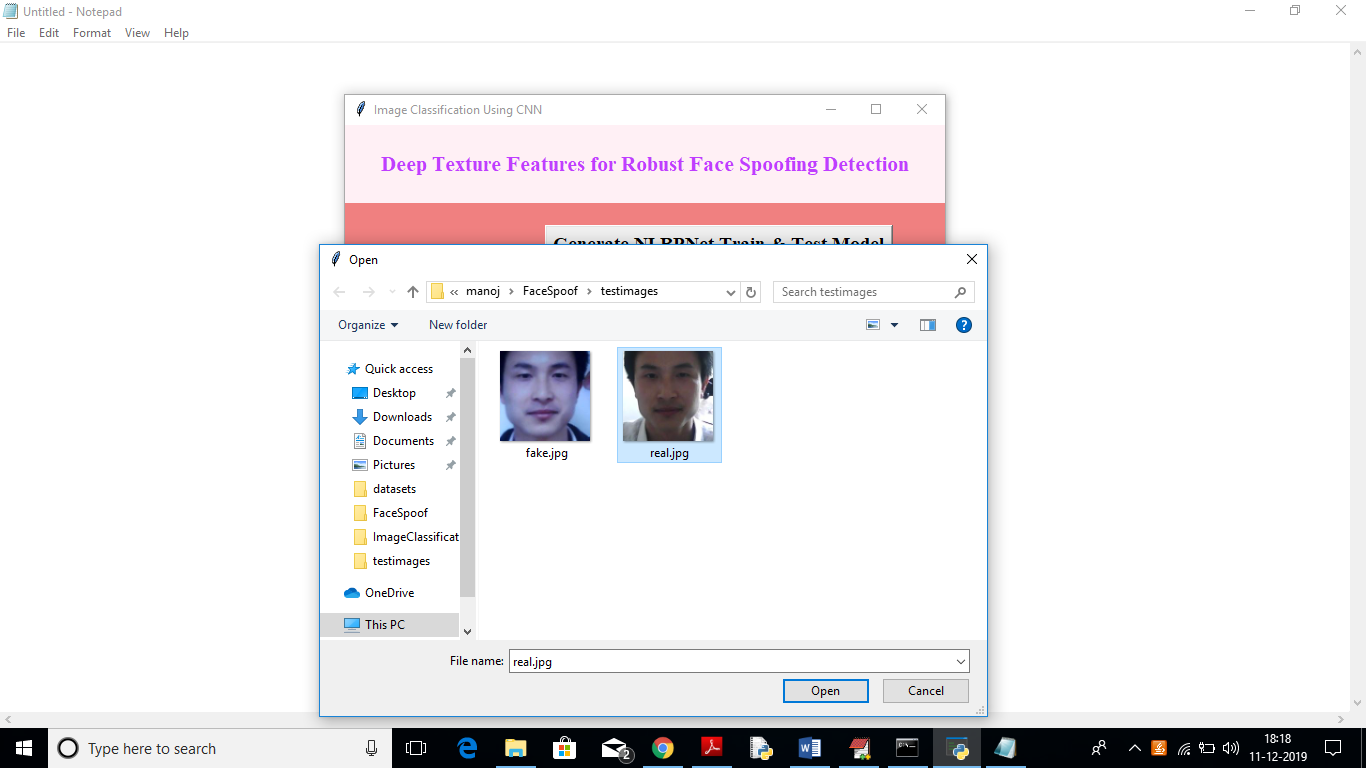
In above screen we can see CNN LBPNET model generated. Now click on ‘Upload Test Image’ button to upload test image



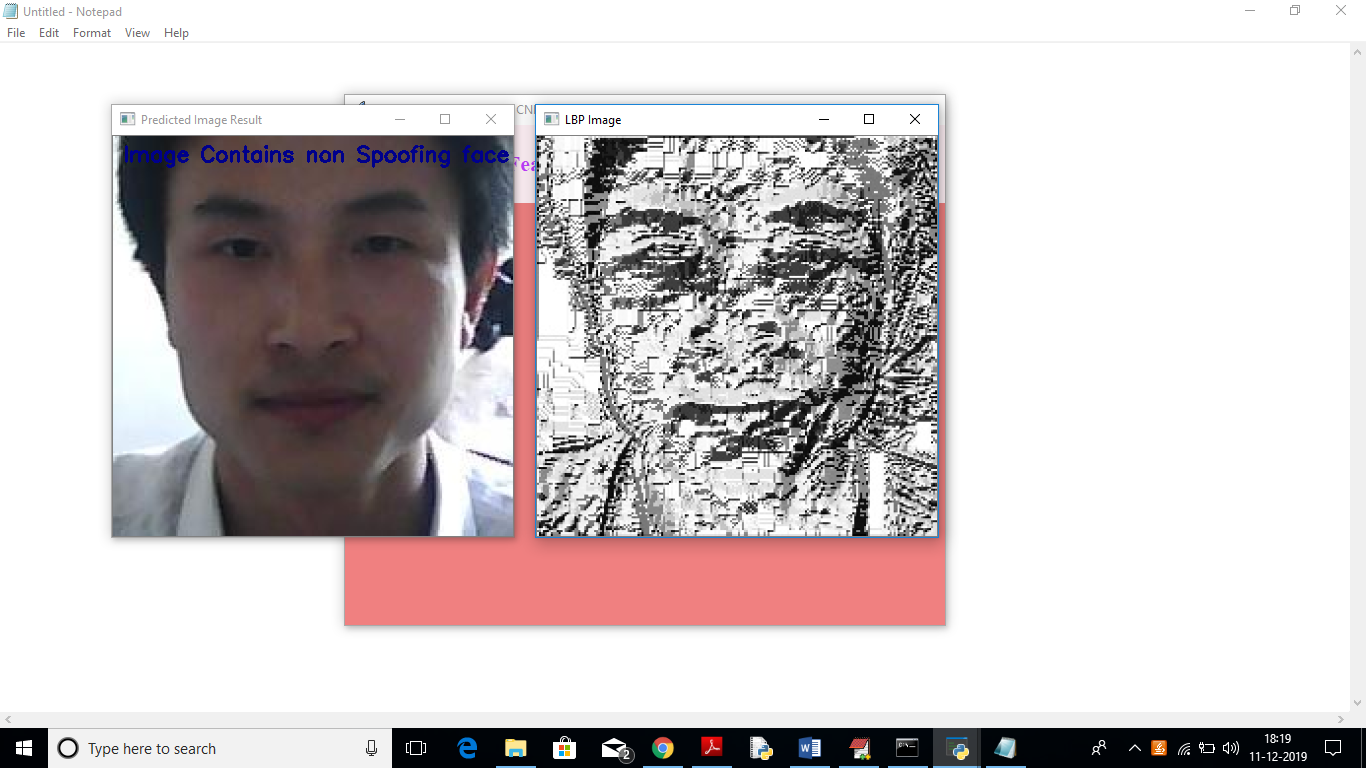
In above screen we can see two faces are there from same person but in different appearances. For simplicity I gave image name as fake and real to test whether application can detect it or not. In above screen I am uploading fake image and then click on ‘Classify Picture In Image’ button to get below result



In above screen application display message on image as it contains spoof face and I am displaying LBP format image also. Now we I will upload real image



Below are the result for uploaded image



In above screen application display message on image as it contains non-spoofing face.

You can use other image also available inside data folder. This data folder contains fake and real faces in separate folders. You can upload from that folder also for testing