Source Code:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 12 21:37:39 2017

@author: Kunal
"""

from PIL import Image
import numpy as np
import math as mt
import random as rn
import os

'''To train the Image Classifier'''
def IC_Train(file_path, h_bin):
    no_train_files = len(os.listdir(file_path))
    model = [['Label', [0 for i in range(h_bin*3)]] for j in range(no_train_files)]
    ind = 0
    for file in os.listdir(file_path):
        im = Image.open(file_path + file)
        im_arr = np.asarray(im)
        row, col, x = im_arr.shape
        file_t = file.split('_')
        c_label = file_t[0]
        model[ind][0] = c_label
        for i in range(row):
            for j in range(col):
                model[ind][1][int((im_arr[i][j][0])/(256/h_bin))] =
model[ind][1][int((im_arr[i][j][0])/(256/h_bin))] + 1
                model[ind][1][int((im_arr[i][j][1])/(256/h_bin)) + h_bin] =
model[ind][1][int((im_arr[i][j][1])/(256/h_bin)) + h_bin] + 1
                model[ind][1][int((im_arr[i][j][2])/(256/h_bin)) + (h_bin*2)] =
model[ind][1][int((im_arr[i][j][2])/(256/h_bin)) + (h_bin*2)] + 1
        if(hist_verify(model[ind][1], row*col)):
            print('The Histogram of Image ' + file + ' is correct.')
        else:
            print('The Histogram of Image ' + file + ' is correct.')
        ind = ind + 1
    print()
    return model

'''To verify that the histogram is correct'''
def hist_verify(hist, no_pixels):
    count = 0
    for i in hist:
        count = count + i
```

```
    if(count/3 == no_pixels):
        return True
    else:
        return False

'''To classify the test images using the Image Classifier'''
def IC_Test(file_path, model):
    h_bin = int(len(model[0][1])/3)
    correct_pred = 0
    wrong_pred = 0
    for file in os.listdir(file_path):
        im = Image.open(file_path + file)
        im_arr = np.asarray(im)
        file_t = file.split('_')
        og_label = file_t[0]
        temp_model = [0 for i in range(h_bin*3)]
        row, col, x = im_arr.shape
        for i in range(row):
            for j in range(col):
                temp_model[int((im_arr[i][j][0])/(256/h_bin))] = temp_model[int((im_arr[i][j][0])/(256/h_bin))]
+ 1
                temp_model[int((im_arr[i][j][1])/(256/h_bin)) + h_bin] =
temp_model[int((im_arr[i][j][1])/(256/h_bin)) + h_bin] + 1
                temp_model[int((im_arr[i][j][2])/(256/h_bin)) + (h_bin*2)] =
temp_model[int((im_arr[i][j][2])/(256/h_bin)) + (h_bin*2)] + 1
        dist = 999999.0
        for mod in model:
            if(eucl_dist(mod[1], temp_model) < dist):
                dist = eucl_dist(mod[1], temp_model)
                test_label = mod[0]
        if(og_label == test_label):
            correct_pred = correct_pred + 1
        else:
            wrong_pred = wrong_pred + 1
        print('Test Image ' + file + ' of class ' + og_label + ' has been assigned to ' + test_label + '.')
    accuracy = (correct_pred/(correct_pred + wrong_pred)) * 100
    print('\nThe Accuracy of the Image Classifier is ' + str(accuracy) + ' when the number of bins is '+
str(h_bin) +'.')

'''To calculate the Eucledian distance'''
def eucl_dist(m1, m2):
    dist = 0.0
    for i in range(len(m1)):
        dist = dist + ((m1[i] - m2[i])**2)
    return mt.sqrt(dist)

'''To train the Pixel classifier'''
def PC_Train(file_path, k):
```

```
    og_im = Image.open(file_path+'sky_train.jpg')
    mask_im = Image.open(file_path+'sky_train_mask_yellow.jpg')
    og_im_arr = np.asarray(og_im)
    row, col, x = og_im_arr.shape
    mask_im_arr = np.asarray(mask_im)
    mask_color = mask_im_arr[0][0]
    sky_set = []
    nonsky_set = []
    for i in range(row):
        for j in range(col):
            if(mask_im_arr[i][j][0] == mask_color[0] and mask_im_arr[i][j][1] == mask_color[1] and
mask_im_arr[i][j][2] == mask_color[2]):
                sky_set.append([int(og_im_arr[i][j][0]), int(og_im_arr[i][j][1]), int(og_im_arr[i][j][2])])
            else:
                nonsky_set.append([int(og_im_arr[i][j][0]), int(og_im_arr[i][j][1]), int(og_im_arr[i][j][2])])
    print('\nThe K-means Algorithm for the sky set starts here: \n')
    sky_words = k_mean(sky_set, k)
    print('\nThe K-means Algorithm for the non-sky set starts here: \n')
    nonsky_words = k_mean(nonsky_set, k)
    return sky_words, nonsky_words

'''To perform K-means clustering to obtain k words'''
def k_mean(ip_set, k):
    no_iter = 1
    converge_flag = 0
    centers = []
    set_len = len(ip_set)
    while(len(centers) != k):
        ran_i = rn.randint(0, set_len-1)
        ran_c = ip_set[ran_i]
        if(ran_c not in centers):
            centers.append(ran_c)
    while(converge_flag == 0):
        converge_flag = 1
        print("Iteration "+str(no_iter))
        clusters = [[] for i in range(len(centers))]
        for i in range(set_len):
            dist = 9999.0
            for k in range(len(centers)):
                c_dist = dist_color(centers[k][0], centers[k][1], centers[k][2], ip_set[i][0], ip_set[i][1],
ip_set[i][2])
                if(c_dist < dist):
                    dist = c_dist
                    c_ind = k
            clusters[c_ind].append([ip_set[i][0], ip_set[i][1], ip_set[i][2]])
        no_new_c = 0
        for i in range(len(clusters)):
            no_pts = len(clusters[i])
```

```
            r = 0; g = 0; b = 0
            for pt in clusters[i]:
                r = r + (int(pt[0])*int(pt[0]))
                g = g + (int(pt[1])*int(pt[1]))
                b = b + (int(pt[2])*int(pt[2]))
            r = r/no_pts; g = g/no_pts; b = b/no_pts
            r = int(mt.sqrt(r)); g = int(mt.sqrt(g)); b = int(mt.sqrt(b))
            new_color = [r, g, b]
            old_color = [centers[i][0], centers[i][1], centers[i][2]]
            if(new_color != old_color):
                no_new_c = no_new_c + 1
                centers[i][0] = r; centers[i][1] = g; centers[i][2] = b
                converge_flag = 0
        print("New centers:", no_new_c)
        print()
        no_iter = no_iter + 1
    return centers

'''To calculate the color distance for K-means'''
def dist_color(r1, g1, b1, r2, g2, b2):
    r_mean = (int(r1) + int(r2)) / 2
    r = int(r1) - int(r2)
    g = int(g1) - int(g2)
    b = int(b1) - int(b2)
    dist = float(mt.sqrt( (((512+r_mean)*r*r)/256) + (4*g*g) + (((767-r_mean)*b*b)/256) ))
    return dist

'''To classify and color pixels according to the Pixxel Classifier'''
def PC_Test(file_path, sky_words, nonsky_words):
    for file in os.listdir(file_path):
        im = Image.open(file_path + file)
        im_arr = np.asarray(im)
        im_cl = np.asarray(im)
        im_cl.setflags(write = 1)
        row, col, x = im_arr.shape
        for i in range(row):
            for j in range(col):
                s_dist = 999999.0
                for k in range(len(sky_words)):
                    if(dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], sky_words[k][0], sky_words[k][1],
sky_words[k][2]) < s_dist):
                        s_dist = dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], sky_words[k][0],
sky_words[k][1], sky_words[k][2])
                ns_dist = 999999.0
                for k in range(len(nonsky_words)):
                    if(dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], nonsky_words[k][0],
nonsky_words[k][1], nonsky_words[k][2]) < ns_dist):
```

```
                ns_dist = dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], nonsky_words[k][0],
nonsky_words[k][1], nonsky_words[k][2])
            if(s_dist < ns_dist):
                im_cl[i][j] = [255, 0, 0]
    op_im = Image.fromarray(im_cl)
    file_temp = file.split('.')
    save_name = file_temp[0]
    op_im.save(save_name+'_output_k'+str(len(sky_words))+'.jpg')
    op_im.show()

def main():
  IC_train_filepath = "./ImClass/Train/"
  h_bin = 10
  c_model = IC_Train(IC_train_filepath, h_bin)
  IC_test_filepath = "./ImClass/Test/"
  IC_Test(IC_test_filepath, c_model)
  PC_train_filepath = "./Sky/Train/"
  k = 10
  sky_words, nonsky_words = PC_Train(PC_train_filepath, k)
  PC_test_filepath = "./Sky/Test/"
  PC_Test(PC_test_filepath, sky_words, nonsky_words)

main()
```

Output Images:

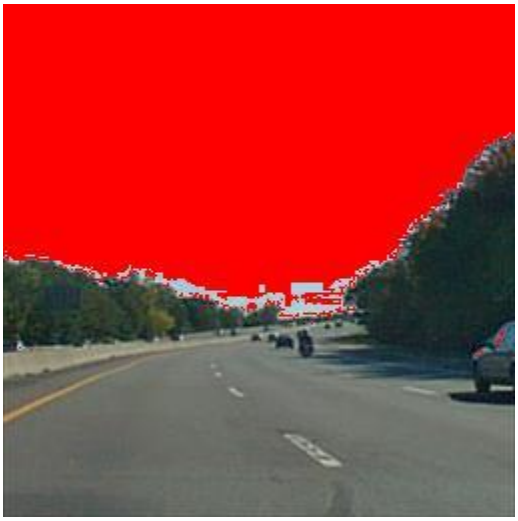
Output of the Image Classifier:

The Histogram of Image coast_train1.jpg is correct.
The Histogram of Image coast_train2.jpg is correct.
The Histogram of Image coast_train3.jpg is correct.
The Histogram of Image coast_train4.jpg is correct.
The Histogram of Image forest_train1.jpg is correct.
The Histogram of Image forest_train2.jpg is correct.
The Histogram of Image forest_train3.jpg is correct.
The Histogram of Image forest_train4.jpg is correct.
The Histogram of Image insidecity_train1.jpg is correct.
The Histogram of Image insidecity_train2.jpg is correct.
The Histogram of Image insidecity_train3.jpg is correct.
The Histogram of Image insidecity_train4.jpg is correct.

Test Image coast_test1.jpg of class coast has been assigned to coast.
Test Image coast_test2.jpg of class coast has been assigned to coast.
Test Image coast_test3.jpg of class coast has been assigned to coast.
Test Image coast_test4.jpg of class coast has been assigned to coast.
Test Image forest_test1.jpg of class forest has been assigned to forest.
Test Image forest_test2.jpg of class forest has been assigned to forest.
Test Image forest_test3.jpg of class forest has been assigned to forest.
Test Image forest_test4.jpg of class forest has been assigned to forest.
Test Image insidecity_test1.jpg of class insidecity has been assigned to forest.
Test Image insidecity_test2.jpg of class insidecity has been assigned to forest.
Test Image insidecity_test3.jpg of class insidecity has been assigned to insidecity.
Test Image insidecity_test4.jpg of class insidecity has been assigned to insidecity.

The Accuracy of the Image Classifier is 83.33333333333334 when the number of bins is 8.

Output Images obtained from Pixel Classifier:



The pixels in the image Sky_test1 colored Red are classified as Sky pixels (k = 10).

The pixels in the image Sky_test2 colored Red are classified as Sky pixels (k = 10).

The pixels in the image Sky_test3 colored Red are classified as Sky pixels (k = 10).

The pixels in the image Sky_test4 colored Red are classified as Sky pixels (k = 10).

Notes:

Image Classification:

For the Training stage of this problem, I am forming a histogram for each image, which is a 24D list in python, along with a label, which indicates the class to which that image belongs, i.e., coast, forest or inside-city. Each octet within the histogram represents the frequencies for each color channel, and thus, there are 3 octets within each histogram to reflect the RGB color channels.

Thus, for each image, every pixel is counted thrice. Thus, the sum of all the 24 numbers within the histogram would be equal to three times the total number of pixels in that image. This is a concept that I am using to verify each histogram for correctness.

For the Classification part, I form a histogram of the test image, and calculate its distance with each of the training image histograms. Accordingly, I assign it the label of the image for which the distance is the smallest.

I got the following accuracies with the corresponding number of bins. I got the best accuracy within the range of 6 – 10 bins.

Bins	2	4	8	16	32	64	128	256
Accuracy	75.0	83.34	83.34	75.0	75.0	75.0	66.67	66.67

Pixel Classification:

For the Training stage, I use the mask image as a reference for the segregation of sky and non-sky pixels and thus, form two disjoint sets, the sum of the number of elements of which should equal the total number of pixels. These sets consist of the RGB values of the pixels. I then use K-means Clustering to obtain a set of distinct visual words, for each class or set of pixels. These words are also RGB triplets. The visuals words for the sky set represent the colors that could possibly come from sky pixels, while the visual words for the non-sky set represent the colors that could possibly come from non-sky pixels.

For the Testing stage, each pixel of every test image is compared with the visual words for the sky set and the non-sky set. Depending on the class to which the word belongs for which the distance is the smallest, the pixel is classified, and if it is a sky pixel, it is colored red, otherwise it is left as it is.

I have attached the output images for different values of k, i.e., 1, 2, 5, 10 and 20. I noticed that a sky pixel may not be marked as one, or a non-sky pixel may be marked as a sky pixel, which can be possible if there are colors in the non-sky region which are similar to those in the sky region. Reducing the value of k helps in improving the region marked as sky, however, also leads to an increase in the false positives, especially in the 3rd test image. It works well if the sky color is close to a shade of blue, because it was trained that way and the sky in the training stage image was distinctly blue. Thus, a pixel is marked as a sky pixel if it is distinctly blue. It fails if there are pixels in the non-sky region with blue-like color shade, which causes a false positive, or if a sky pixel is of color, which is not so blue. I noticed that sky pixels which were bluish grey or close to grey in color were not marked as sky pixels.