```
from PIL import Image
import random
import os

def ImageClassification(bins):
    training_histograms = []
    divisor = 256 / bins
    #loop through the 3 folders
    for i in range(3):
        if i == 0:
            path = "C:\workspace\Visual Recognition\VisualRecognition\ImClass\CityTraining"
        if i == 1:
            path = "C:\workspace\Visual Recognition\VisualRecognition\ImClass\CoastTraining"
        if i == 2:
            path = "C:\workspace\Visual Recognition\VisualRecognition\ImClass\ForestTraining"
        #loop through images in folder
        for image_path in os.listdir(path):
            # create the full input path and read the file
            input_path = os.path.join(path, image_path)
            img = Image.open(input_path)
            pixels = img.load()
            #first row is R, second is G, third is B
            histogram = [[0 for u in range(bins)] for s in range(3)]
            img_width = img.size[0]
            img_height = img.size[1]
            for x in range(img_width):
                for y in range(img_height):
                    r_val = int(pixels[x, y][0] // divisor)
                    g_val = int(pixels[x, y][1] // divisor)
                    b_val = int(pixels[x, y][2] // divisor)
                    histogram[0][r_val] += 1
                    histogram[1][g_val] += 1
                    histogram[2][b_val] += 1
            #add histogram to training_histograms along with image_path
            training_histograms.append([image_path, histogram])

    #loop through test images
    path = "C:\workspace\Visual Recognition\VisualRecognition\ImClass\TestImages"
    num_wrong = 0
    for image_path in os.listdir(path):
        # create the full input path and read the file
        input_path = os.path.join(path, image_path)
        img = Image.open(input_path)
        pixels = img.load()
        #first row is R, second is G, third is B
        histogram = [[0 for u in range(bins)] for s in range(3)]
        img_width = img.size[0]
        img_height = img.size[1]
```

```
        for x in range(img_width):
            for y in range(img_height):
                r_val = int(pixels[x, y][0] // divisor)
                g_val = int(pixels[x, y][1] // divisor)
                b_val = int(pixels[x, y][2] // divisor)
                histogram[0][r_val] += 1
                histogram[1][g_val] += 1
                histogram[2][b_val] += 1
        #find nearest fit with euclidean distance
        min_dist = 1000000
        min_h = 0
        #loop through list of histograms
        for h in range(len(training_histograms)):
            e_sum = 0
            #loop through color rows
            for r in range(3):
                #loop through values in color row
                for v in range(len(training_histograms[h][1][r])):
                    e_sum += (histogram[r][v] - training_histograms[h][1][r][v])**(2)
            e_sum = e_sum**(1/2)
            if(e_sum < min_dist):
                min_dist = e_sum
                min_h = h
        print(image_path + " has been assigned to " + training_histograms[min_h][0] + ".")
        if(image_path[0] != training_histograms[min_h][0][0]):
            num_wrong += 1
    print("Test with " + str(bins) + " bins had " + str(num_wrong) + " incorrect assignments for
an accuracy " +
          "of " + ('%.2f' % (((12-num_wrong)/12)*100)) + " percent.")

def PixelClassification():
    path_train = "C:\workspace\Visual Recognition\VisualRecognition\sky\sky_train.jpg"
    path_train_pink = "C:\workspace\Visual
Recognition\VisualRecognition\sky\sky_train_pink.jpg"

    img_train = Image.open(path_train)
    img_train_pink = Image.open(path_train_pink)

    pixels_train = img_train.load()
    pixels_train_pink = img_train_pink.load()

    img_width = img_train.size[0]
    img_height = img_train.size[1]

    sky = []
    non_sky = []
    for x in range(img_width):
        for y in range(img_height):
```

```
            if(pixels_train_pink[x, y] == (255, 0, 232)):
                sky.append([x, y])
            else:
                non_sky.append([x, y])

    sky_clusters, mean_rgb_sky = k_means(pixels_train, sky, 10)
    non_sky_clusters, mean_rgb_nonsky = k_means(pixels_train, non_sky, 10)

    path_tests = "C:\\workspace\\Visual Recognition\\VisualRecognition\\sky\\test"
    for image_path in os.listdir(path_tests):
        # create the full input path and read the file
        input_path = os.path.join(path_tests, image_path)
        img_test = Image.open(input_path)
        pixels_test = img_test.load()

        img_width = img_test.size[0]
        img_height = img_test.size[1]

        sky_color = (0, 255, 0)
        for x in range(img_width):
            for y in range(img_height):
                #[dist, 0 or 1] 0 for sky, 1 for nonsky
                min_dist = [1000000, 0]
                for j in range(10):
                    dist_sky = ((pixels_test[x, y][0] - mean_rgb_sky[j][0])**2
                            + (pixels_test[x, y][1] - mean_rgb_sky[j][1])**2
                            + (pixels_test[x, y][2] - mean_rgb_sky[j][2])**2)**(1/2)
                    dist_nonsky = ((pixels_test[x, y][0] - mean_rgb_nonsky[j][0])**2
                            + (pixels_test[x, y][1] - mean_rgb_nonsky[j][1])**2
                            + (pixels_test[x, y][2] - mean_rgb_nonsky[j][2])**2)**(1/2)
                    if(dist_sky < dist_nonsky):
                        if(dist_sky < min_dist[0]):
                            min_dist[0] = dist_sky
                            min_dist[1] = 0
                    else:
                        if(dist_nonsky < min_dist[0]):
                            min_dist[0] = dist_nonsky
                            min_dist[1] = 1
                #if pixel is closest to sky visual word, color it
                if(min_dist[1] == 0):
                    pixels_test[x, y] = sky_color
        #save image
        img_test.save(image_path[:len(image_path)-4] + "_green.jpg")

def k_means(pixels, points, k):
    #pick random point from list of points either in sky or non_sky
    rand_centers = random.sample(points, k)
```

```
    centers = []
    for i in range(k):
        #format: width, height, red, green, blue
        centers.append([pixels[rand_centers[i][0], rand_centers[i][1]][0],
                        pixels[rand_centers[i][0], rand_centers[i][1]][1],
                        pixels[rand_centers[i][0], rand_centers[i][1]][2]])

    done = False
    while(not done):
        cluster_list, mean_rgb_list = cluster_formation(centers, pixels, points, k)

        new_dists = []
        #tbd
        threshold = 5
        for j in range(k):
            dist = ((centers[j][0] - mean_rgb_list[j][0])**2
                    + (centers[j][1] - mean_rgb_list[j][1])**2
                    + (centers[j][2] - mean_rgb_list[j][2])**2)**(1/2)
            new_dists.append(dist)

        outside_threshold = False
        for x in new_dists:
            if(x > threshold):
                outside_threshold = True

        if(outside_threshold):
            centers = mean_rgb_list
        else:
            cluster_list, mean_rgb_list = cluster_formation(centers, pixels, points, k)
            done = True

    return cluster_list, mean_rgb_list

def cluster_formation(centers, pixels, points, k):
    #iterate through points, finding which center is closest and creating clusters
    cluster_list = []
    for i in range(k):
        cluster_list.append([])

    #mean rgb for each cluster
    mean_rgb_list = [[0 for i in range(3)] for j in range(k)]

    #iterating through all pixels in the image
    for p in points:
        #[dist, cluster_num]
        min_dist = [1000000000, 0]
        #iterating through the 10 centers to find which is closest to each pixel
        for a in range(10):
```

```
            #calculating euclidean distance from point to center in terms of color values
            dist = ((centers[a][0] - pixels[p[0], p[1]][0])**2
                    + (centers[a][1] - pixels[p[0], p[1]][1])**2
                    + (centers[a][2] - pixels[p[0], p[1]][2])**2)**(1/2)
            if(dist == min_dist[0]):
                min_dist[0] = dist
                min_dist[1] = random.choice([min_dist[1], a])
            if(dist < min_dist[0]):
                min_dist[0] = dist
                min_dist[1] = a
        #populating clusters
        cluster_list[min_dist[1]].append((p[0], p[1]))
        mean_rgb_list[min_dist[1]][0] += pixels[p[0], p[1]][0]
        mean_rgb_list[min_dist[1]][1] += pixels[p[0], p[1]][1]
        mean_rgb_list[min_dist[1]][2] += pixels[p[0], p[1]][2]

    for t in range(k):
        total_vals = len(cluster_list[t])
        mean_rgb_list[t][0] = round(mean_rgb_list[t][0] / total_vals)
        mean_rgb_list[t][1] = round(mean_rgb_list[t][1] / total_vals)
        mean_rgb_list[t][2] = round(mean_rgb_list[t][2] / total_vals)

    return cluster_list, mean_rgb_list

def main():

    #first with 8 bins
    ImageClassification(8)
    #second with 16 bins
    ImageClassification(16)
    #third with 4 bins
    ImageClassification(4)

    PixelClassification()

if __name__ == "__main__":
    main()
```

coast_test1.jpg has been assigned to coast_train3.jpg.
coast_test2.jpg has been assigned to coast_train2.jpg.
coast_test3.jpg has been assigned to coast_train1.jpg.
coast_test4.jpg has been assigned to coast_train1.jpg.
forest_test1.jpg has been assigned to forest_train3.jpg.
forest_test2.jpg has been assigned to forest_train2.jpg.
forest_test3.jpg has been assigned to forest_train4.jpg.
forest_test4.jpg has been assigned to forest_train2.jpg.
insidecity_test1.jpg has been assigned to forest_train4.jpg.
insidecity_test2.jpg has been assigned to forest_train1.jpg.
insidecity_test3.jpg has been assigned to insidecity_train3.jpg.
insidecity_test4.jpg has been assigned to insidecity_train4.jpg.
Test with 8 bins had 2 incorrect assignments for an accuracy of 83.33 percent.
coast_test1.jpg has been assigned to coast_train3.jpg.
coast_test2.jpg has been assigned to coast_train3.jpg.
coast_test3.jpg has been assigned to coast_train3.jpg.
coast_test4.jpg has been assigned to coast_train2.jpg.
forest_test1.jpg has been assigned to forest_train3.jpg.
forest_test2.jpg has been assigned to forest_train2.jpg.
forest_test3.jpg has been assigned to forest_train2.jpg.
forest_test4.jpg has been assigned to forest_train2.jpg.
insidecity_test1.jpg has been assigned to forest_train4.jpg.
insidecity_test2.jpg has been assigned to forest_train1.jpg.
insidecity_test3.jpg has been assigned to forest_train2.jpg.
insidecity_test4.jpg has been assigned to insidecity_train4.jpg.
Test with 16 bins had 3 incorrect assignments for an accuracy of 75.00 percent.
coast_test1.jpg has been assigned to coast_train2.jpg.
coast_test2.jpg has been assigned to coast_train2.jpg.
coast_test3.jpg has been assigned to coast_train1.jpg.
coast_test4.jpg has been assigned to coast_train2.jpg.
forest_test1.jpg has been assigned to forest_train3.jpg.
forest_test2.jpg has been assigned to forest_train2.jpg.
forest_test3.jpg has been assigned to forest_train2.jpg.
forest_test4.jpg has been assigned to forest_train2.jpg.
insidecity_test1.jpg has been assigned to forest_train4.jpg.
insidecity_test2.jpg has been assigned to insidecity_train3.jpg.
insidecity_test3.jpg has been assigned to insidecity_train3.jpg.
insidecity_test4.jpg has been assigned to coast_train1.jpg.
Test with 4 bins had 2 incorrect assignments for an accuracy of 83.33 percent.

The program is hardcoded for a specific folder structure, so if you want to run it for images on your
machine you'll have to change the path variables on lines 11, 13, 15, 38, 79, 80, and 103. In general,
in whatever folder you have the __init__.py file you should copy and paste the folders (ImClass and sky)
that I've included in the zip as they already have most of the structure needed. For instance, if the path is this in the code: "C:\workspace\Visual Recognition\VisualRecognition\sky\sky_train_pink.jpg"
you'll just have to change the "C:\workspace\Visual Recognition\VisualRecognition\" to whatever
your path is and the rest should be the same.

The ImageClassification output is included in a .txt file and it runs with 8 bins, 16 bins, and 4 bins.

REPORT:
Problem 1: The accuracy for 8 bins was 83.33 percent, for 16 bins it was 75 percent, and for 4 bins it was also 83.33 percent. Thus it seems that the number of bins does not contribute to the accuracy of the algorithm.

Problem 2: The main issue with the algorithm for problem 2 is that it only compares color and does not factor in the actual location of pixels in the image. For example in sky_test3, there are parts near the bottom of the image that were similar enough in color to be classified as sky pixels but since they are in the bottom of the image with a bunch of non-sky pixels around them they obviously cannot be sky. The opposite problem happens in sky_test4 where some of the parts of the sky are similar in color to parts of the ground and are put in a cluster with nonsky pixels despite being surrounded by other sky pixels. The algorithm works when the sky and nonsky are relatively distinct in their rgb values. An improvement would be to incorporate the location of the pixel in the k-means algorithm so that if a sky pixel is surrounded by non-sky pixels or vice-versa, the algorithm will know to change that pixel to nonsky or sky respectively.