Kunal Dhaimade - CS 558 Optional Problem 1

Source Code:

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 28 20:20:37 2017

@author: Kunal
"""

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

'''To create a Histogram representation (descriptor) for each of the images in the dataset'''
def createHistogram(file_path, h_bin):
    no_train_files = len(os.listdir(file_path))
    model = [[j, [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
        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(histVerify(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 histVerify(hist, no_pixels):
    count = 0
    for i in hist:
        count = count + i
    if(count/3 == no_pixels):
        return True
    else:
        return False
```

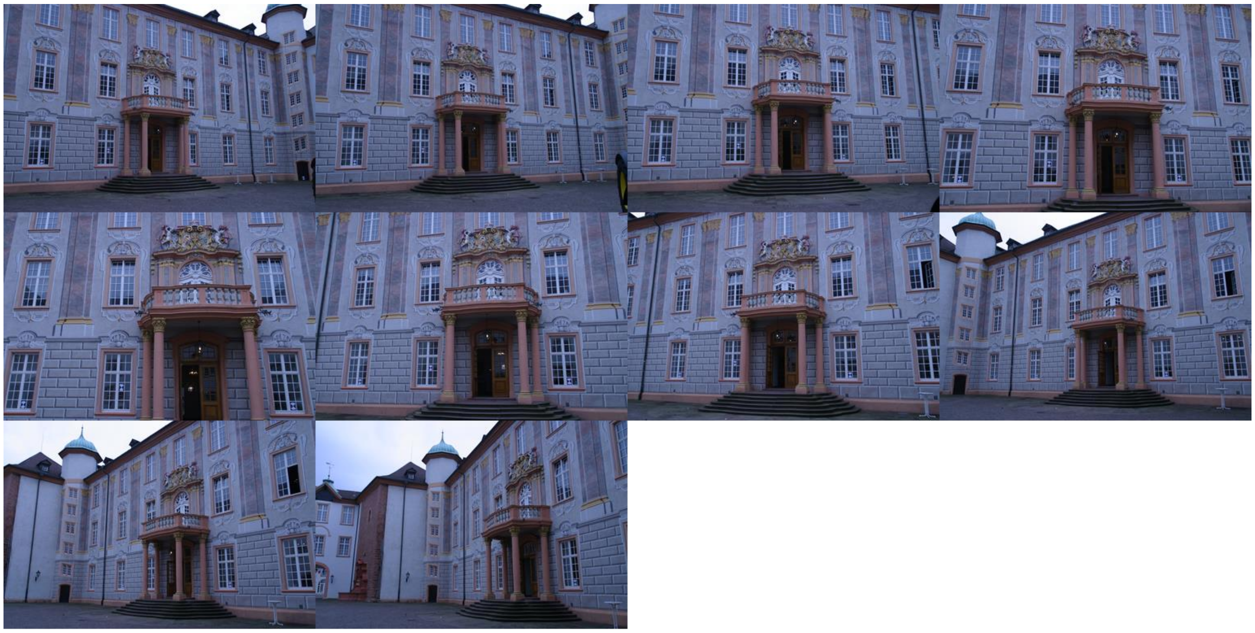
Kunal Dhaimade - CS 558 Optional Problem 1

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

''''''
def formCluster(histogram, t_dist, kernel_bw):
    means = []
    clusters = []
    for i in histogram:
        x = i[1]
        converge_flag = 0
        while(converge_flag == 0):
            neighbors  = []
            converge_flag = 1
            for im in histogram:
                if(euclDist(x, im[1]) < t_dist):
                    neighbors.append(im)
            if(neighbors != []):
                mean = meanShift(neighbors, x, kernel_bw)
            if(mean != x):
                x = mean
                converge_flag = 0
            else:
                if(mean not in means):
                    means.append(mean)
                    clusters.append([i[0]])
                else:
                    ind = means.index(mean)
                    clusters[ind].append(i[0])
    return clusters, means

'''To calculate the Mean Histogram using the Mean-Shift factor'''
def meanShift(neighbors, x, k_bw):
    size = len(neighbors[0][1])
    mean = [0 for i in range(size)]
    for i in range(size):
        num = float(0)
        den = float(0)
        shift = float(0)
        for n in neighbors:
            dist = euclDist(n[1], x)
            weight = gaussianKernel(dist, k_bw)
            num = num + (weight * n[1][i])
            den = den + weight
        shift = num/den
        mean[i] = round(shift)
    return mean

'''To calculate the weight using the Gaussian kernel'''
def gaussianKernel(distance, bandwidth):
    val = (1/(bandwidth*mt.sqrt(2*mt.pi))) * np.exp(-0.5*((distance / bandwidth))**2)
    return val

def drawClusters(file_path, clusters, save_name):
    c_no = 1
    for c in clusters:
        no_e = len(c)
        f = os.listdir(file_path)
        im_r, im_c, x = np.asarray(Image.open(file_path + f[0])).shape
        scale = mt.ceil(mt.sqrt(no_e))
        col_r = im_r * scale
        col_c = im_c * scale
        op = Image.new('RGB', (col_c, col_r), color = 'White')
        op = np.asarray(op)
        op.setflags(write = 1)
        init_i = 0
        init_j = 0
        s_fac = 0
        for i in c:
            im = np.asarray(Image.open(file_path + f[i]))
            t_r, t_c, t_x = im.shape
            for i in range(t_r):
                for j in range(t_c):
                    op[init_i + i][init_j + j] = im[i][j]
            init_j = init_j + im_c
            s_fac = s_fac + 1
            if(s_fac == scale):
                init_i = init_i + im_r
                init_j = 0
                s_fac = 0
        op = Image.fromarray(op)
        op.save(save_name + '_Cluster_' + str(c_no) + '.png')
        c_no = c_no + 1

def main():
    Images1_filepath = "./Images_1/"
    Images2_filepath = "./Images_2/"
    h_bin = 4
    histogram1 = createHistogram(Images1_filepath, h_bin)
    histogram2 = createHistogram(Images2_filepath, h_bin)
    t_dist = 50000.0
    kernel_bw = 30000.0
    clusters1, means1 = formCluster(histogram1, t_dist, kernel_bw)
    saveName1 = 'Images_1'
    drawClusters(Images1_filepath, clusters1, saveName1)
    clusters2, means2 = formCluster(histogram2, t_dist, kernel_bw)
    saveName2 = 'Images_2'
    drawClusters(Images2_filepath, clusters2, saveName2)

main()
```

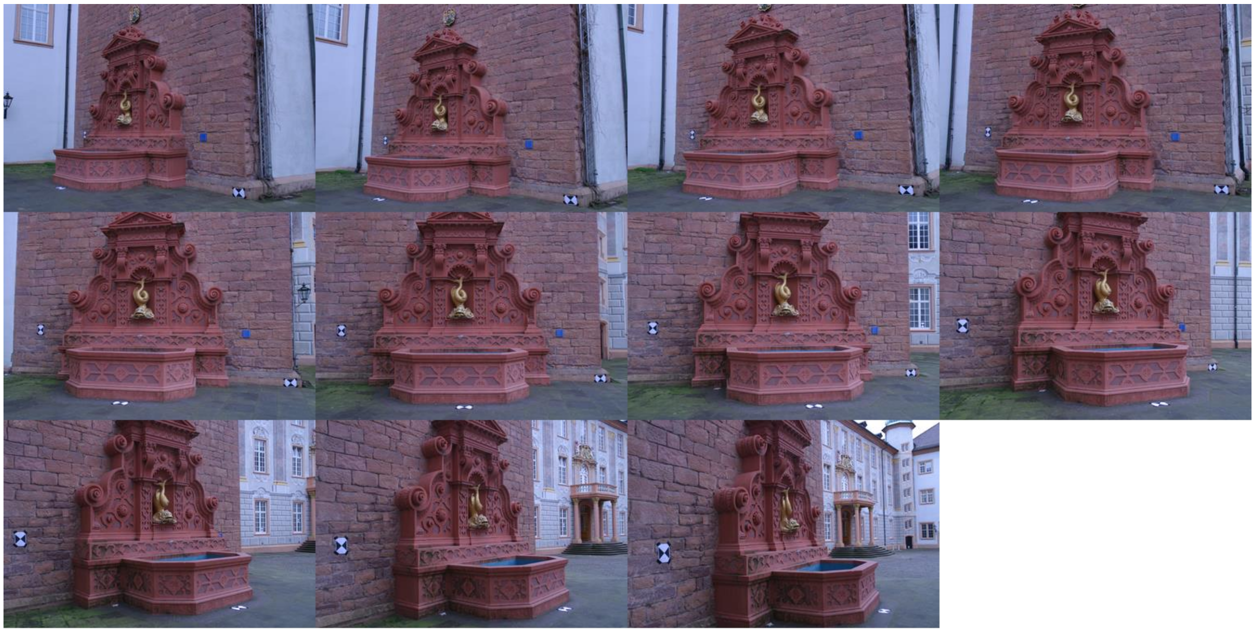
Kunal Dhaimade - CS 558 Optional Problem 1

Output Images:

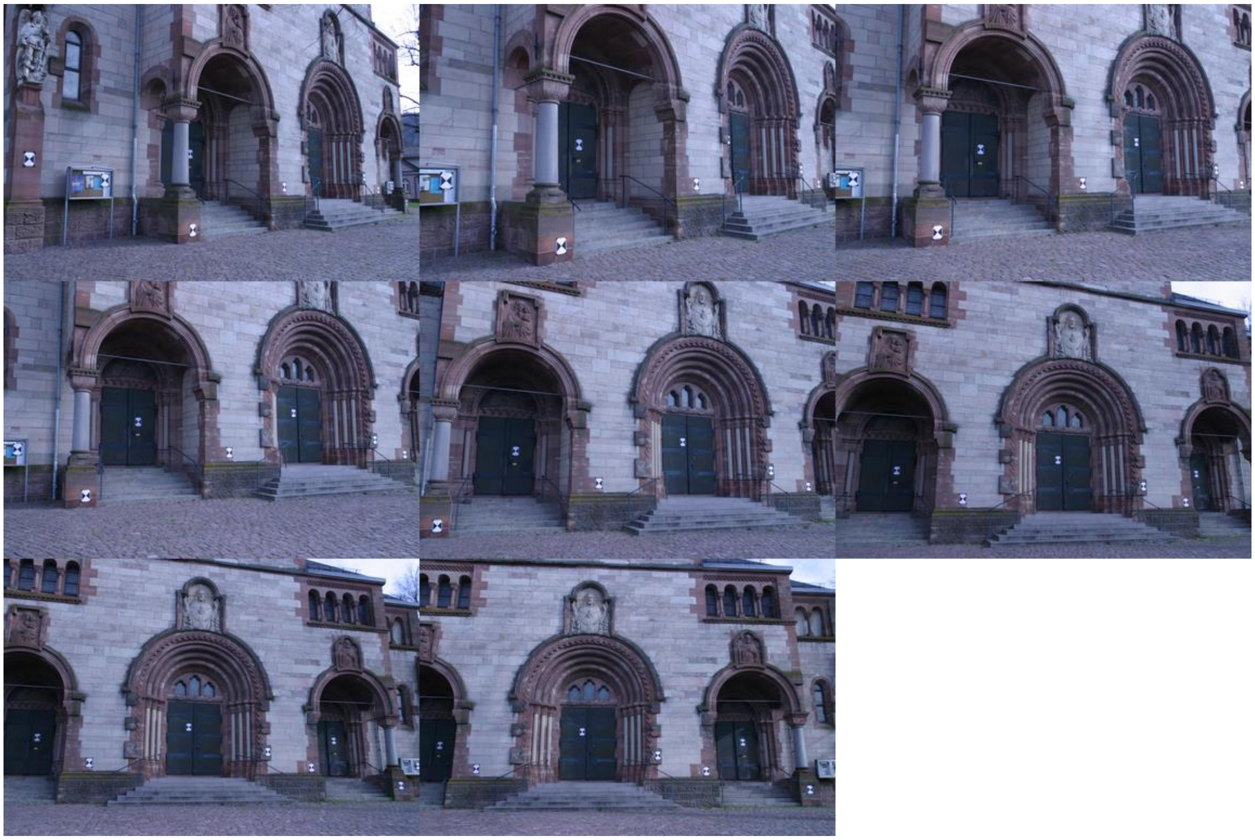
Output for Data Set 1 (29 images):



Cluster 1 (10 images)



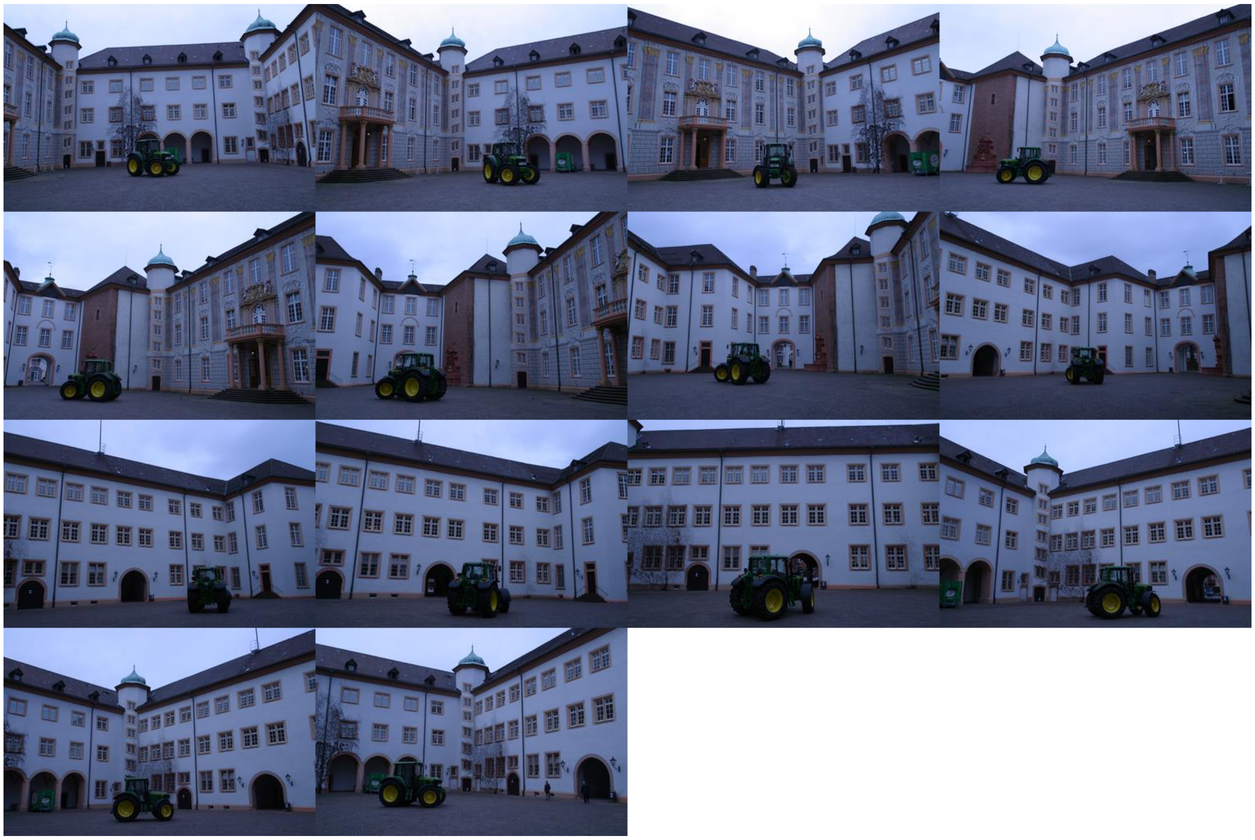
Cluster 2 (11 images)



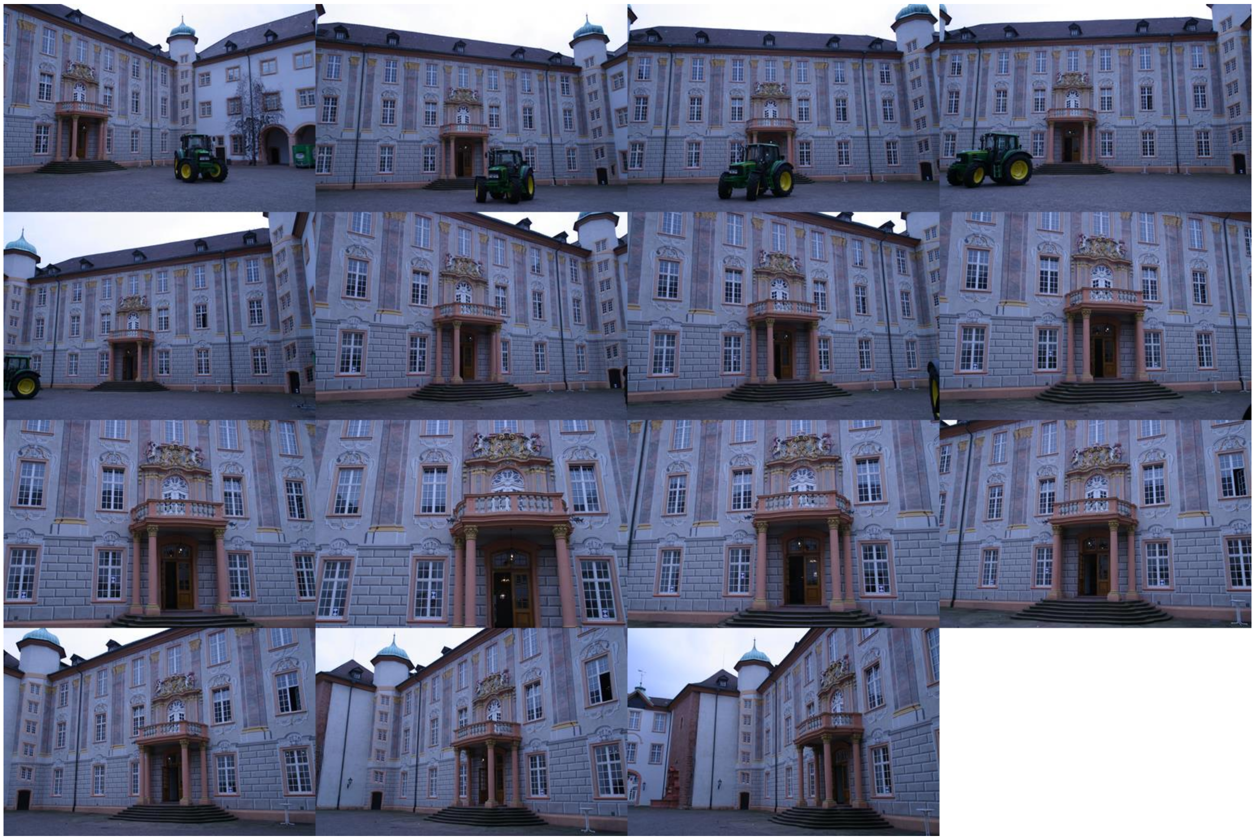
Cluster 3 (8 images)

Kunal Dhaimade - CS 558 Optional Problem 1

Output for Data Set 2 (29 images):



Cluster 1 (14 images)



Cluster 2 (15 images)

Kunal Dhaimade - CS 558 Optional Problem 1

Notes:

Image Data Association:

I have implemented the Mean Shift algorithm as instructed for the purpose of forming the Clusters. I have used the Gaussian kernel for the same. Each image is considered as a point for the Mean Shift problem, and is represented by its corresponding histogram. I have used a 4 bin histogram as the image descriptor. The primary parameters for the algorithm are the Threshold Distance and the Kernel Bandwidth.

To tune the parameters, I established a baseline for the distance between the histograms belonging to the same group, and then for the histograms outside of the group. This helped me determine a limit to ensure the formation of a cluster as perfect as possible. To ensure good clustering, I experimented the algorithm with different histogram bins (8, 16, 32) and also by varying the input parameters. I displayed the output as a collage of all the images belonging to one cluster.

For the first set of images (entry, fountain, herz-jesu), I obtained 3 clusters and the clusters I obtained were perfect for a threshold distance of 50000.0 and a kernel bandwidth of 30000.0. For the second set of images (castle, entry), I obtained 2 clusters having a few images wrongly assigned, with the same parameters. I have resized the images while maintaining the same aspect ratio, for faster formation of histograms, and to create the output cluster collage images. The folders Images_1 and Images_2 correspond to the first and the second aggregate image data set respectively.