Vincent Lee
I pledge my honor that I have abided by the Stevens Honor System.

meanShift.py is where all the code is and runs parameterized.

```
if __name__ == "__main__":
    print('Data Set 1:')
    cluster("data1/*.jpg",1880000)
    print('Data Set 2:')
    cluster("data2/*.jpg",1920000)
```

The algorithm is essentially to get the descriptor of each image. I used a 24 dimensional vector which is the composition of 8 bin histograms for each color channel of an image. I used skimage exposure's method to generate the histogram as running my own from hw4 was not fast/optimized to be done for so many and large images. After which I call meanshift which assigns each descriptor to a mean signifying which cluster it belongs to. Meanshift returns the index of which clusters the images correspond to. After which I print which images correspond to each cluster.

```
def cluster(path,maxDistance):
    data = glob.glob(path)
    descriptors = []
    images = []
    # iterate through images
    for path in data:
        image = img_as_float(imageio.imread(path))
        histogramR,_ = exposure.histogram(image[:,:,0],nbins=8)
        histogramG,_ = exposure.histogram(image[:,:,1],nbins=8)
        histogramB,_ = exposure.histogram(image[:,:,2],nbins=8)
        descriptor = generateFeatures([histogramR,histogramG,histogramB])
        descriptors.append(descriptor)
        images.append((images,path))
    # 1880000 for 3 cluster
    # 1800000 for 5 clusters but good seperation
    indexValues = meanShift(descriptors,maxDistance)
    numIndex = len(set(indexValues))
    print(numIndex, ' Clusters')
    imageSets = [[] for _ in range(numIndex)]
    for i in range(len(images)):
        targetIndex = indexValues[i]
        imageSets[targetIndex].append(images[i])
    for i in range(numIndex):
        print('Cluster ', i, ': ', end='')
        for j in range(len(imageSets[i])):
            print(imageSets[i][j][1],', ', end='')
        print('')
```

Mean shift takes the descriptors copies it to a new list. It then iterates through each descriptor (which we will call the target descriptor) and then compares that target descriptor with every other descriptor. If a descriptor is within the maxDistance from the target descriptor then we add it to a surroundings list. This is the “kernel” where every element within the kernel is of weight 1. It then averages the surrounding list and makes the average the new target descriptor. It then iterates through all descriptors again until the target descriptor no longer changes.

```
def meanShift(descriptors, maxDistance):
    means = numpy.copy(descriptors)
    for i in range(len(means)):
        target = means[i]
        surroundings = []
        converged = False
        while not converged:
            for temp in descriptors:
                # "kernel" function
                # if the distance to the next descriptor is less than the maxDistance you can include it
                # with the calculation of the mean with weight 1
                distance = numpy.linalg.norm(target-temp)
                if distance <= maxDistance:
                    surroundings.append(temp)
            center = numpy.mean(surroundings,axis=0)
            if numpy.linalg.norm(target-center) == 0:
                converged = True
            target = center
            surroundings = []
        means[i] = target
    ret = indexDescriptors(means)
    return ret
```

Index descriptors takes the converged descriptors and returns a list of indices where every descriptor with the same value is assigned the same index. It uses comparisons with tuples and gets unique elements by doing converting the list of tuples into sets and then back into lists.

```
# assign descriptors to the index of center they converged to
def indexDescriptors(means):
    means = means.tolist()
    swap = []
    for temp in means:
        swap.append(tuple(temp))
    means = copy.deepcopy(swap)
    uniqueValues = list(set(swap))
    ret = []
    # iterate through each descriptor
    for i in range(len(means)):
        target = means[i]
        # if the descriptor equals the center assign the index value
        for j in range(len(uniqueValues)):
            center = uniqueValues[j]
            if target == center:
                ret.append(j)
                break
    return ret
```

Generate features takes in the list of histograms and concatenates them. Since each of ours was histograms of size 8 the total is of size 24.

```python
# concatenate each histogram to a 24 dimension list
def generateFeatures(histograms):
    ret = []
    for histogram in histograms:
        ret.extend(histogram)
    return ret
```

The biggest issue in this implementation is the tuning of the maximum distance. After playing with the values, the ones currently set to run will generate 3 clusters and 2 clusters respectively. The data set 1 has 3 different types of images and the data set 2 has 2 different as specified in the assignment description.

Here are the results:

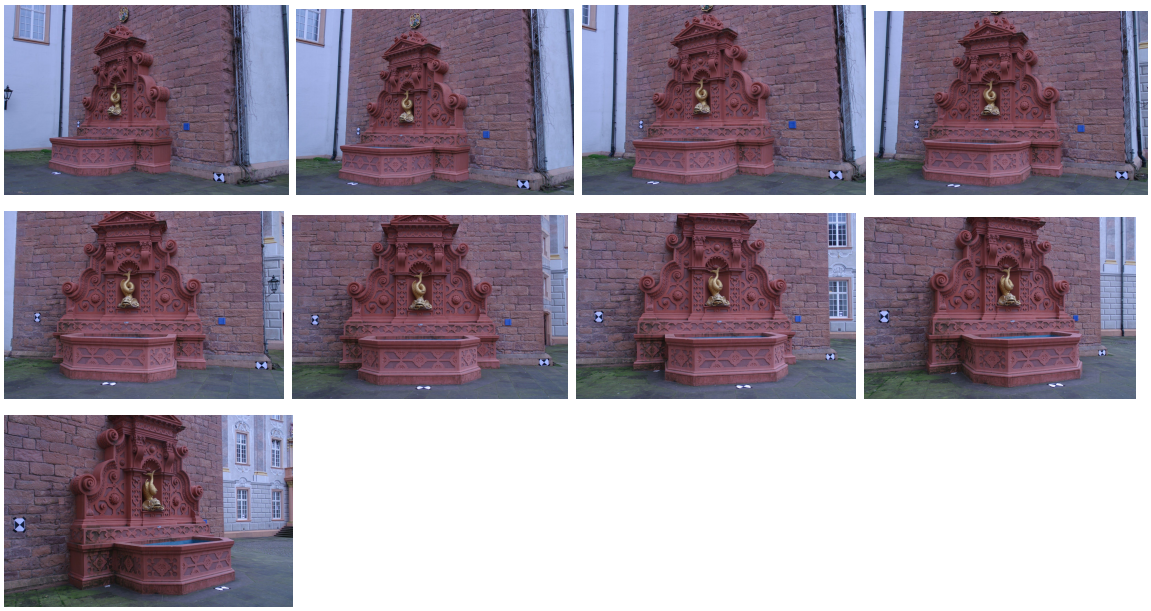
```
3  Clusters
Cluster  0 : data1\entry-P100000.jpg , data1\entry-P100001.jpg , data1\entry-P100002.jpg , data1\entry-P100003.jpg , data1\entry-P1000
04.jpg , data1\entry-P100005.jpg , data1\entry-P100006.jpg , data1\entry-P100007.jpg , data1\entry-P100008.jpg , data1\entry-P100009.j
pg , data1\Herz-Jesus-P80003.jpg ,
Cluster  1 : data1\fountain0000.jpg , data1\fountain0001.jpg , data1\fountain0002.jpg , data1\fountain0003.jpg , data1\fountain0004.jp
g , data1\fountain0005.jpg , data1\fountain0006.jpg , data1\fountain0007.jpg , data1\fountain0008.jpg ,
Cluster  2 : data1\fountain0009.jpg , data1\fountain0010.jpg , data1\Herz-Jesus-P80000.jpg , data1\Herz-Jesus-P80001.jpg , data1\Herz-
Jesus-P80002.jpg , data1\Herz-Jesus-P80004.jpg , data1\Herz-Jesus-P80005.jpg , data1\Herz-Jesus-P80006.jpg , data1\Herz-Jesus-P80007.j
pg ,
Data Set 2:
2  Clusters
Cluster  0 : data2\castle-P190000.jpg , data2\castle-P190001.jpg , data2\castle-P190002.jpg , data2\castle-P190003.jpg , data2\castle-
P190004.jpg , data2\castle-P190005.jpg , data2\castle-P190006.jpg , data2\castle-P190007.jpg , data2\castle-P190008.jpg , data2\castle
-P190009.jpg , data2\castle-P190010.jpg , data2\castle-P190011.jpg , data2\castle-P190012.jpg , data2\castle-P190013.jpg , data2\castl
e-P190014.jpg , data2\castle-P190015.jpg , data2\castle-P190016.jpg , data2\castle-P190017.jpg , data2\castle-P190018.jpg , data2\entr
y-P100009.jpg ,
Cluster  1 : data2\entry-P100000.jpg , data2\entry-P100001.jpg , data2\entry-P100002.jpg , data2\entry-P100003.jpg , data2\entry-P1000
04.jpg , data2\entry-P100005.jpg , data2\entry-P100006.jpg , data2\entry-P100007.jpg , data2\entry-P100008.jpg ,
PS C:\Users\vince\Desktop\cs558\ExtraCredit\EC1>
```

Data set 1 was trained with maximum distance of 1880000 and data set 2 was trained with maximum distance of 1920000. There was a lot playing with this hyperparameter in order to generate the number of clusters I wanted. I chose the number of clusters I wanted to target corresponding to the different image sets.

When you visualize it.

Dataset 1:
Cluster 0:

Cluster 1:



Cluster 2:

Dataset 2:

Cluster 0:








Cluster 1: