Assignment 5

Johnnie Oldfield

October 2019

1 Problem Definition

The purpose of this assignment is to implement the K-means algorithm on images. Reducing the number of colors in the image. There are around 16 million possible colors for a single pixel, many of which can be considered unnecessary as the human eye can only perceive so many. Every image is defined by pixels that are based on three values (red, green, and blue). To perform K-means on an image would mean getting K set of color clusters. Then going through the image and replace the pixels with the color of the closest pixel. The main experiment will be using 8 online images that can be access through URL.

2 Decisions

To start I define all the images I decided to use with imread. The images are then normalised in a range of [0, 1] and stored into a list of images. See listing 1.

Listing 1: Getting and normalising images

```
waterfall = imread ('https://....jpg')
...
waterfall = waterfall / 255.0
..
images = [waterfall, nature, ...]
```

I decided to use pandas data frames for manipulating images. It made looking at the image data much easier. Images are eventually changed back into a list.

Listing 2: K-means.

```
K = [2, 4, 8]

for im in images:
  plot_images = []
  plot_images.append(im)
  im_df = image_to_df(im)
  for k in K:
    df = im_df.copy(deep=True)
    centroids = get_centroids(im, k)
    df = assignment(df, centroids)
    new_df, costs, new_centroids = maximization(df, centroids)
    new_image = replace_pixels(new_df, new_centroids)
    plot_images.append(df_to_im(new_image, im))
```

This nested for loop goes through the list of of images the the list of k. Plot_images list will hold the original as well as the other 3 images once they have been compressed with each k.

Listing 3: Image to dataframe and vice versa.

```
def image_to_df(im):
  im = np.reshape(im, (im.shape[0] * im.shape[1], im.shape[2]))
  df = pd.DataFrame.from_records(im)
  df.columns = ['r', 'g', 'b']
  return df

def df_to_im(df, im):
  df2 = df.filter(['r','g','b'], axis=1)
  df2 = df2.to_numpy()
  new_im = np.reshape(df2, (im.shape[0], im.shape[1], ...
      im.shape[2]))
  return new_im
```

Listing 3 shows the functions I use for converting images and data frames. I reshaped it into a 2-d array and gave it columns for each color channel of the pixel. Other columns are added to the data frame and they are filtered out when converting back to an image.

Listing 4: Getting and Assigning Centroids

```python
def get_centroids(im, k):
  r, c, clr  = im.shape
  centroids = {i+1: im[(np.random.randint(0, r - 1)),
        (np.random.randint(0, c - 1))] for i in range(k)}
  return r, c, centroids

def assignment(df, centroids):
  for i in centroids.keys():
    df['distance_from_{}'.format(i)] = (np.sqrt(
                    ((df['r'] - centroids[i][0])**2) +
                    ((df['g'] - centroids[i][1])**2) +
                    ((df['b'] - centroids[i][2])**2)))
  centroid_distance_cols = ['distance_from_{}'.format(i)
                    for i in centroids.keys()]
  df['closest'] = df.loc[:,
                    centroid_distance_cols].idxmin(axis=1)
  df['closest'] = df['closest'].map
                    (lambda x: int(x.lstrip('distance_from_')))
  return df
```

I get random centroids from the image using get_centroids. They are random pixels grabbed and contains the rgb channels. The assignment function uses the data frame and centroids to find the closest centroid to each pixel. They are then updated; see listing 5.

Listing 5: Update function

```python
def update(df, centroids):
  old_centroids = copy.deepcopy(centroids)
  for i in centroids.keys():
    if(True not in df['closest'] == i):
      continue
    else:
      updated_centroids[i] = np.mean(df[df['closest'] == i])
  return centroids
```

Listing 6: Maximixation and replacing pxiels

```
def maximization(df, centroids):
  it = 0
  costs = []
  centroid_distance_cols = ['distance_from_{}'.format(i) ...
      for i in centroids.keys()]
  while it < 25:
    # E - step
    centroids = update(df, centroids)
    # M - step
    df = assignment(df, centroids)
    cost = df.loc[:,  centroid_distance_cols].min(axis=1).sum()
    if(len(costs)==0 or costs[-1]!=cost):
      costs.append(cost)
    else:
      break
    it+=1
    return df, costs, centroids

def replace_pixels(df, centroids):
  for ix, row in df.iterrows():
    color = centroids[row['closest']]
    df.at[ix,'r'] = color[0]
    df.at[ix,'g'] = color[1]
    df.at[ix,'b'] = color[2]
  return df
```

3 Evaluation

For the assignment I have a set amount of K's, ranging 2, 4, and 8. These k represent the number of centroids for the algorithm (in other words the color). I then ran 8 images through k-means with each k to see the results. As ecpected; each image displays amounts of colors equal to k.

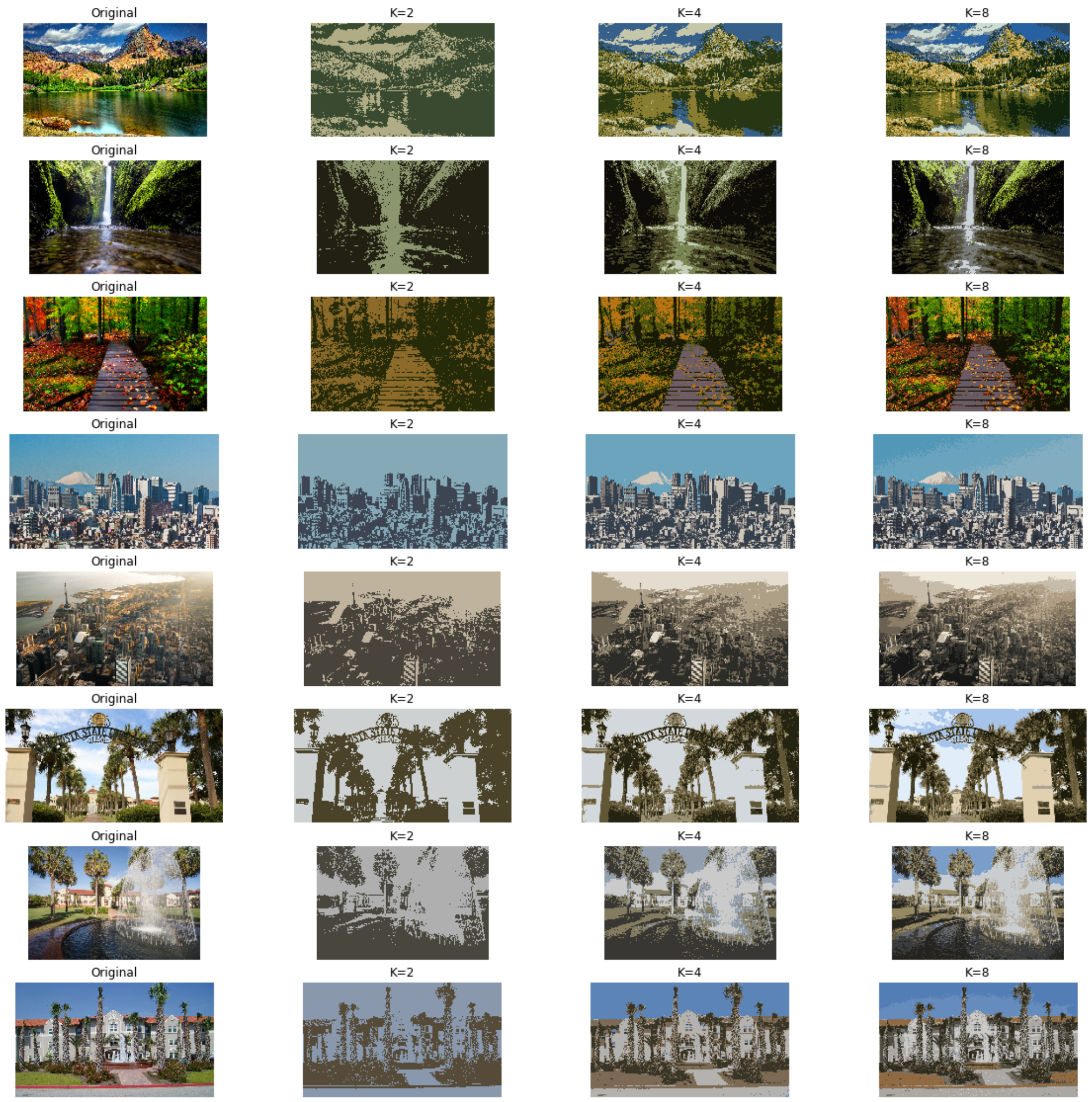

Figure 1: Problem Solution

4 What I Learned

I learned a lot on the k-means algorithm and it's affect on image compression. Also, I was unaware how easy it is to manipulate images. The extensive use of data frames also gave me a better understanding on how they can be utilized.

5 Future Work

The k-means implementation takes a long time to run due the the large number of data frames. It could be more efficient if lists were used instead. Extra experimentation was not conduced so there is more to learn with this problem.