Project 4 - Color Image Compression Using Unsupervised Learning (Clustering)

Adrian Cross

November 25, 2019

across19@vols.utk.edu
Machine learning
COSC 522

1 Abstract

In this project an image has been input and the RGB value for each image has been gained. Using various clustering algorithms the original possible RGB variations has been reduced to various different numbers by using different clusters amounts. The results of these different algorithms and cluster amounts have been compared by directly looking at the images and by computing percentage differences between original and final color values. As expected the quality of images decreased with cluster number. It was also found that the quantifiable percentage error rates were similar but there was a vast difference in the actual images.

2 Introduction

An image is made up of rows and columns with a pixel making up each point in the image. Each pixel is made up of a variety of factors, including a color value for red, green and blue. For a full color image each color value has a total possible number of 2^{24}. On some computers the total number of colors are reduced to 2^8 or 256 colors. By reducing the number of colors the storage size of the image is reduced. This number can be further reduced which, to 2^7,2^6, etc., reduces the storage size of the image further. In this project the image shown in figure 1 has been reduced to various different sizes via various different algorithms and the result of this reduction has been compared.

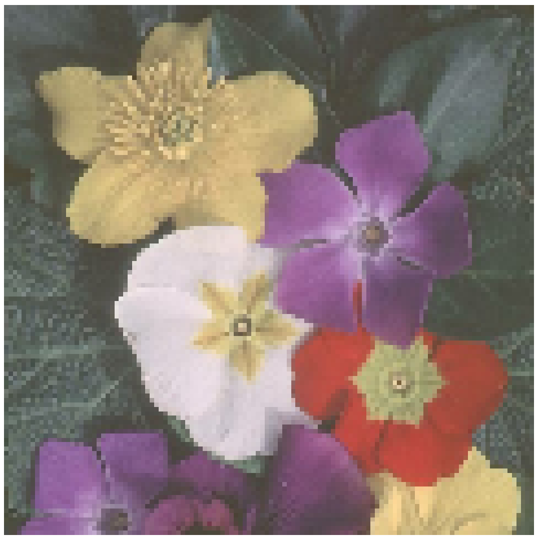

Figure 1: Original input image

The reduction in colors is done via clustering algorithms. These algorithms start with the clusters assigned in specific places and then move the clusters dependent on the input data. The RGB coordinates of these clusters is one of the new color values which each data point can take. For example if a cluster point was at $(R, G, B) = (154, 124, 123)$ then the pixels with RGB coordinates closest to that cluster, and not closer to any other cluster, will be assigned a new RGB value of $(154, 124, 123)$. The output image has been compared both via by directly looking at the output images, with reduced colors, and by looking at the percentage difference between the old color value and the new color value, outlined in section 3.3.

3 Experiments and Results

3.1 Input

In section 2 the input image was introduced (figure 1). By organizing this image into rows and columns, with the corresponding number of pixels in the rows and columns, the image can be converted to a data frame of size $120 * 120 = 14400$ with each data point being one pixel with a color value for red, green and blue (RGB). By plotting these RGB values on a graph we can see the distribution of colors in this image, as shown in image 2 by the blue data points.

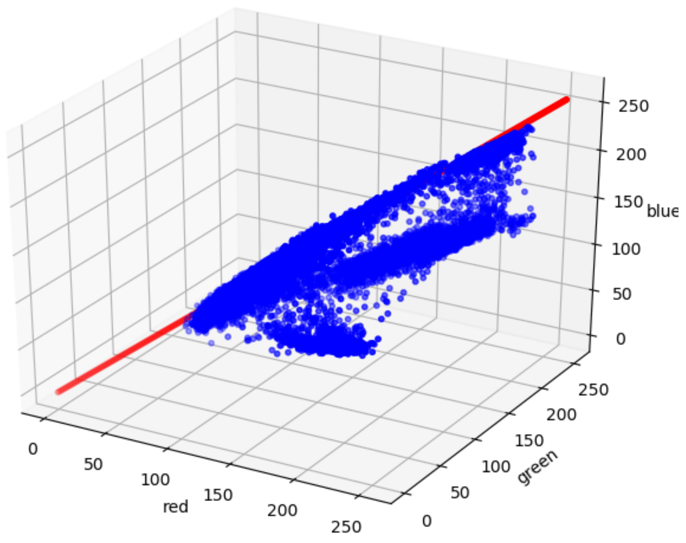

Figure 2: Pixel RGB values (blue) with 258 clusters distribution (red)

From this you can see the pixels have an RGB value between 0 and 256, concentrated around the center of the graph. This indicates the overall color of the image, for example if the image was mostly shades of red this image would show clustering which is biased towards the high red RGB value.

3.2 Initializing Clusters

After the RGB values have been inputted the initial cluster positions need to be defined. This can be done via a variety of means, such as random selection, but in this case the start positions have simply been defined as a straight line starting at $(R, G, B) = 0, 0, 0$ up to $(256, 256, 256)$ in even RGB steps [Steinbach et al., 2000]. The even steps are determined by the number of clusters used in each analysis. In this project a value of $256, 128, 64, 32, 16, 8, 4and2$ has been chosen for the cluster numbers. Therefore, for 256 clusters, the step inbetween each RGB value will simply be 1, as shown in figure 2 by the red colored line.

3.3 Data Comparison

In order to facilitate the comparison of the output image, two metrics have been used. The first is a simple visual test which is done by outputting the images with the new pixel color values. From this you can see how much the image changes with each compression amount. The second metric attempts to quantify the error between the source image and the output image. This is done by taking the input pixel RGB value and finding the percentage difference to the new RGB value, as determined by the cluster center, as shown in equation 1.

$$\%diff = \frac{(RGB(actual) - RGB(Cluster))}{RGB(actual)} * 100 \tag{1}$$

This percentage difference is then plotted against number of clusters to see the difference in the trend for different number of clusters.

3.4 K-Means

The first algorithm tested is the k-means algorithm. This algorithm works by [University of Tennessee]:

1. Begin with an arbitrary assignment of samples to clusters or begin with an arbitrary set of cluster centers and assign samples to nearest clusters
2. Compute the sample mean of each cluster
3. Reassign each sample to the cluster with the nearest mean
4. If the classification of all samples has not changed, stop; else go to step 2.

   figure 3 shows how these cluster centers have moved from their original position in figure 2.

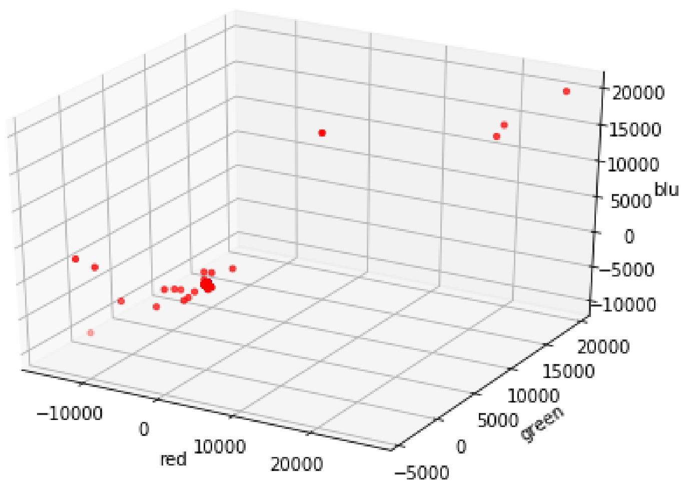

Figure 3: Final cluster position for k-means algorithm

By completing this algorithm over several iterations the final images are found, as shown in figure 9. By computing the percentage difference outlined in 3.3 the different percentage differences for each cluster number has been found and plotted in figure 4.

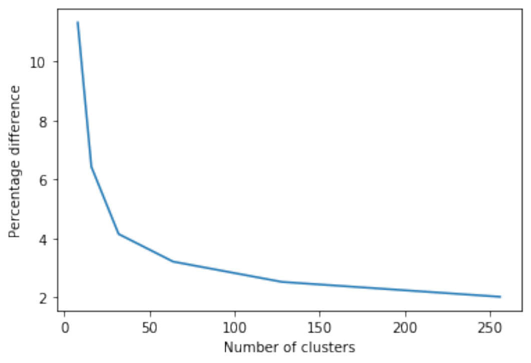

Figure 4: Quality reduction with number of clusters for k-means algorithm

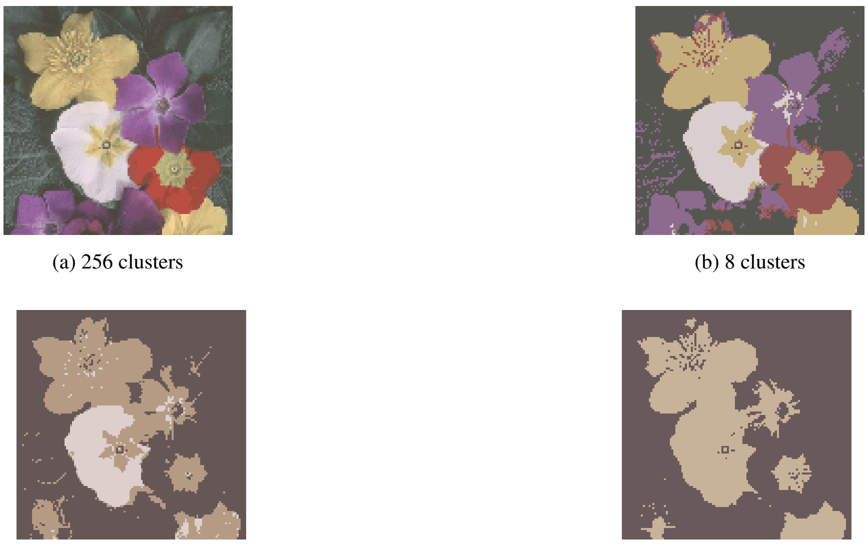
(a) 256 clusters (b) 8 clusters (c) 4 clusters (d) 2 clusters

Figure 5: Image quality reduction for k-means algorithm for different number of clusters

3.5 Winner Take All

The winner-take-all algorithm starts the same way as the k-means algorithm by first initializing the cluster centers. After this each pixel x finds the nearest cluster center which is then defined as the winner. The winning cluster center ω_a is then modified using equation 2.

$$\omega_a^{new} = \omega_a^{old} + \epsilon(x - \omega_a^{old}) \tag{2}$$

Where ϵ is the learning parameter which is approximately equal to ϵ 0.01. By completing this algorithm over several iterations the final images are found, as shown in figure 9. By computing the percentage difference outlined in 6 the different percentage differences for each cluster number has been found and plotted in figure 6.

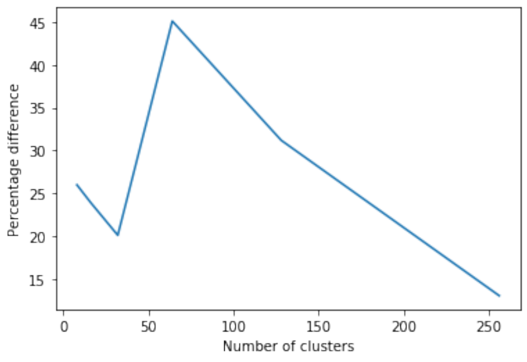

Figure 6: Quality reduction with number of clusters for winner take all algorithm

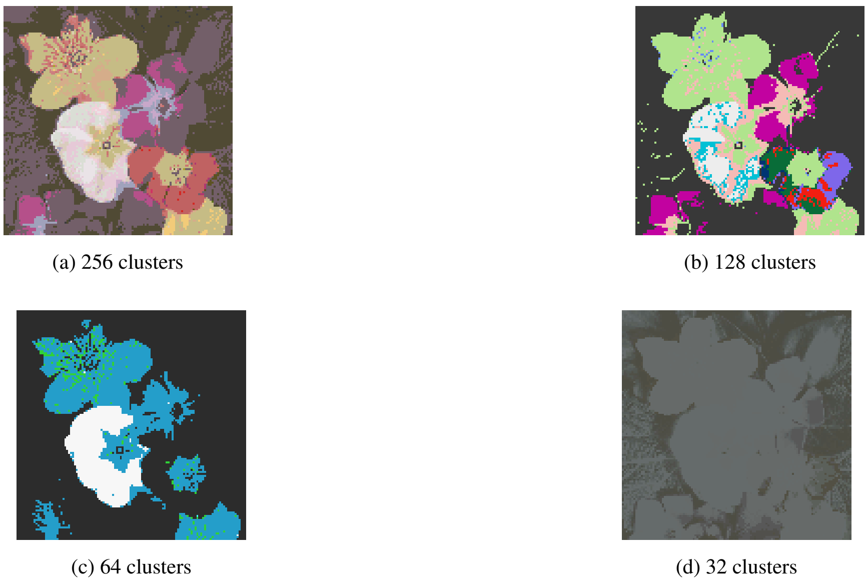

Figure 7: Image quality reduction for winner take all algorithm for different number of clusters

3.6 Kohonen

The Kohonen feature map (NN) is a modification of the winner take all algorithm. It works by assuming a problem dependent topological distance exists between a pair of cluster centers. This means that when the winning cluster center is updated so are its neighbours, dependent on this topological distance. Therefore the algorithm works the same way as winner take all, except equation 2 is modified so that it becomes equation 3.

$$\omega_r^{k+1} = \omega_r^k + \epsilon(k) * \Phi(k)(x - \omega_r^k) \tag{3}$$

Where ω_r are the cluster centers and epsilon is constantly adjusted so that as the number of iterations increases the learning rate decreases as it is becoming more stable, as shown in equation 4

$$\epsilon(k) = \epsilon_{max} * (\frac{\epsilon_{min}}{\epsilon_{max}})^{k/k_{max}} \tag{4}$$

and the topological Φ distance is defined by equation 5

$$\Phi(k) = exp(-\frac{||g_{\omega_r} - frac\epsilon_{min}||^2}{2\sigma^2}) \tag{5}$$

Where g_{ω_r} is the coordinate of the cluster centers and $g_{\omega_{winner}}$ is the coordinate of the winner.

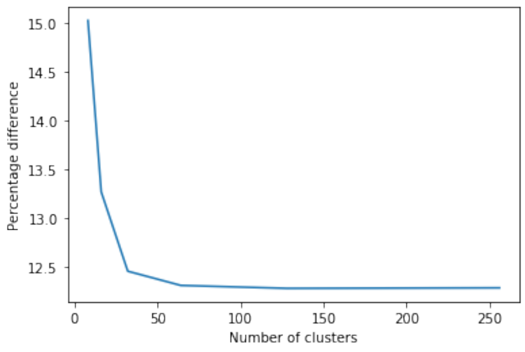

Figure 8: Quality reduction with number of clusters for Kohonen algorithm

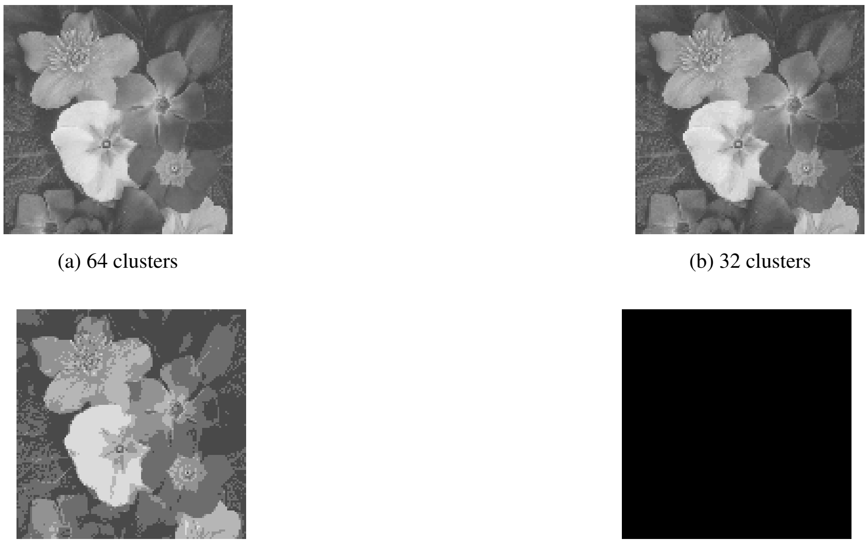
(a) 64 clusters (b) 32 clusters (c) 8 clusters (d) 2 clusters

Figure 9: Image quality reduction for Kohonen algorithm for different number of clusters

4 Conclusion

From the difference in images you can see a stark difference in the way the different algorithms alter the images. In the k-means algorithm the image quality is decreased and the final image is much different than for the winner take all algorithm, however there is still an overall decrease in quality.

While the images are graphically different, the actual reduction in quality, shown by the quantification of these errors using percentage differences, is very similar across each method. Therefore it can be concluded that while each algorithm does reduce both the storage size and percentage error of the image in similar ways, the actual visual quality is changed in different ways. Further study could be done to investigate this further, possibly by using image recognition tools, such as neural networks, in order to quantify the visual differences of the images between different number of clusters, as well as different algorithms.

A Code

A.1 Algorithm File

```
#reads in image and outputs dataframe for clusters and pixels
import sys
import math
import timeit
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from skimage import io

#custom modules
import find_closest_cluster
import plot_rgb
import k_means
import winner_take_all
import fig_io

def run_k_means(n_clusters,pixel_data):
    count_max=10
    clusters= fig_io.initialize_clusters(n_clusters)
    clusters=k_means.initialize(clusters.copy())
    pixel_data_comp=pixel_data
    counter=0
    limit_reached=False
    while(limit_reached==False):
        counter+=1
        pixel_data_comp=pixel_data

        #reset
        clusters=k_means.reset_clusters(clusters.copy())
        clusters.reset_index(drop=True,inplace=True)
        pixel_data=find_closest_cluster.reset_closest_cluster(pixel_data.copy())
        pixel_data.reset_index(drop=True,inplace=True)

        #algorithm
        pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
        clusters=k_means.sample_mean(pixel_data,clusters.copy())
        clusters=k_means.reassign_cluster_pos(clusters.copy())

        #stop conditions
        if pixel_data_comp.equals(pixel_data)==True:
            limit_reached=True
            print(’algorithm has converged stopping....’)
        print(’counter=’+str(counter))
        if counter==count_max:
            limit_reached=True
            print(’algorithm has reached ’+str(counter)+’ iterations stopping....’)


    print(’completed for ’+str(n_clusters)+’ clusters’)
    pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
    pixel_data=find_closest_cluster.update_pixels_to_closest_cluster(pixel_data.copy(),clusters.copy())

    plot_rgb.plot_rgb_cluster_pix(clusters.copy(),pixel_data,’k_means_’+str(n_clusters))
    plot_rgb.save_file(clusters.copy(),pixel_data.copy(),’k_means_’+str(n_clusters))




def run_winner_take_all(n_clusters,pixel_data,learning_parameter):
    count_max=5
    clusters=fig_io.initialize_clusters(n_clusters)
    clusters=winner_take_all.initialize(clusters.copy())
    pixel_data_comp=pixel_data
    counter=0
    limit_reached=False

    while(limit_reached==False):
        counter+=1
        pixel_data_comp=pixel_data

        #reset
        clusters=winner_take_all.reset_clusters(clusters.copy())
        clusters.reset_index(drop=True,inplace=True)
        pixel_data=find_closest_cluster.reset_closest_cluster(pixel_data.copy())
        pixel_data.reset_index(drop=True,inplace=True)

        #algorithm
        pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
        clusters=winner_take_all.pixel_shift(pixel_data.copy(),clusters,learning_parameter)

        #stop conditions
        if pixel_data_comp.equals(pixel_data)==True:
            limit_reached=True
            print(’algorithm has converged stopping....’)
        print(’counter=’+str(counter))
        if counter==count_max:
            limit_reached=True
            print(’algorithm has reached ’+str(counter)+’ iterations stopping....’)

    print(’completed for ’+str(n_clusters)+’ clusters’)
    pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
```

```python
    pixel_data=find_closest_cluster.update_pixels_to_closest_cluster(pixel_data.copy(),clusters.copy())

    plot_rgb.plot_rgb_cluster_pix(clusters.copy(),pixel_data,'wta_'+str(n_clusters))
    plot_rgb.save_file(clusters.copy(),pixel_data.copy(),'wta_'+str(n_clusters))




def run_kohonen(n_clusters,pixel_data,eps_max,eps_min):
    count_max=5
    clusters=fig_io.initialize_clusters(n_clusters)
    clusters=winner_take_all.initialize(clusters.copy())
    pixel_data_comp=pixel_data
    counter=0
    limit_reached=False

    while(limit_reached==False):
        counter+=1
        pixel_data_comp=pixel_data

        #reset
        clusters=winner_take_all.reset_clusters(clusters.copy())
        clusters.reset_index(drop=True,inplace=True)
        pixel_data=find_closest_cluster.reset_closest_cluster(pixel_data.copy())
        pixel_data.reset_index(drop=True,inplace=True)

        #algorithm
        pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
        winner_take_all.kohonen(pixel_data=pixel_data.copy(),clusters=clusters.copy(),epsilon_max=eps_max,epsilon_min=eps_min,k=counter,k_max=count_max)

        #stop conditions
        print('counter='+str(counter))
        if counter==count_max:
            limit_reached=True
            print('algorithm has reached '+str(counter)+' iterations stopping....')

    print('completed for '+str(n_clusters)+' clusters')
    pixel_data=find_closest_cluster.find_all_closest_cluster(pixel_data.copy(),clusters.copy())
    pixel_data=find_closest_cluster.update_pixels_to_closest_cluster(pixel_data.copy(),clusters.copy())

    plot_rgb.plot_rgb_cluster_pix(clusters.copy(),pixel_data,'kohonen_'+str(n_clusters))
    plot_rgb.save_file(clusters.copy(),pixel_data.copy(),'kohonen_'+str(n_clusters))



pixel_data=fig_io.read_in_fig('flowersm.ppm')
for n_clusters in [256,128,64,32,16,8,4,2]:
   # run_k_means(n_clusters,pixel_data)
    #run_winner_take_all(n_clusters,pixel_data,0.05)
    run_kohonen(n_clusters,pixel_data,2,0.5)
```

A.2 Image Output

```python
#output_images to images folder
import sys
import math
import timeit
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scipy.misc as smp
from PIL import Image
from mpl_toolkits.mplot3d import Axes3D
from skimage import io

```

```python
#custom modules
import find_closest_cluster
import plot_rgb
import k_means
import fig_io


for i in [256,128,64,32,8,4,2]:
    pixel_data = pd.read_csv("data/k_means_"+str(i)+"_pixel_data.csv") #k_mean
    fig_io.read_out_fig(pixel_data.copy(),'flowersm.ppm','k_means_'+str(i)+'clusters')

    pixel_data = pd.read_csv("data/wta_"+str(i)+"_pixel_data.csv") #wta
    fig_io.read_out_fig(pixel_data.copy(),'flowersm.ppm','wta_'+str(i)+'clusters')

    pixel_data = pd.read_csv("data/kohonen_"+str(i)+"_pixel_data.csv") #kohonen
    fig_io.read_out_fig(pixel_data.copy(),'flowersm.ppm','kohonen_'+str(i)+'clusters')
```

A.3 Evaluate Algorithms

```python
#evaluates data
import sys
import math
import timeit
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scipy.misc as smp
from PIL import Image
from mpl_toolkits.mplot3d import Axes3D
from skimage import io

#custom modules
import find_closest_cluster
import plot_rgb
import k_means
import fig_io

def r_perc_diff(pixel_data): #returns the mean percentage difference
    perc_diff=(pixel_data['red']-pixel_data['r_cluster'])/pixel_data['red']
    perc_diff=abs(perc_diff*100)
    perc_diff=perc_diff.mean()
    return perc_diff

def g_perc_diff(pixel_data): #returns the mean percentage difference
    perc_diff=(pixel_data['green']-pixel_data['g_cluster'])/pixel_data['green']
    perc_diff=abs(perc_diff*100)
    perc_diff=perc_diff.mean()
    return perc_diff

def b_perc_diff(pixel_data): #returns the mean percentage difference
    perc_diff=(pixel_data['blue']-pixel_data['b_cluster'])/pixel_data['blue']
    perc_diff=abs(perc_diff*100)
    perc_diff=perc_diff.mean()
    return perc_diff

def output_graphs(algorithm):
    cluster_no=[256,128,64,32,16,8]
    perc_diff_r=[]
    perc_diff_g=[]
    perc_diff_b=[]

    for i in cluster_no:
        pixel_data = pd.read_csv("data/"+algorithm+"_"+str(i)+"_pixel_data.csv")
        perc_diff_r.append(r_perc_diff(pixel_data))
        perc_diff_g.append(g_perc_diff(pixel_data))
        perc_diff_b.append(b_perc_diff(pixel_data))
    perc_diff=[(perc_diff_r+perc_diff_g+perc_diff_b)/3 for perc_diff_r,perc_diff_g,perc_diff_b
     in zip(perc_diff_r,perc_diff_g,perc_diff_b)]
    plot_rgb.plot_perc_diff(cluster_no,perc_diff_r,'red',algorithm)
    plot_rgb.plot_perc_diff(cluster_no,perc_diff_g,'green',algorithm)
    plot_rgb.plot_perc_diff(cluster_no,perc_diff_b,'blue',algorithm)
    plot_rgb.plot_perc_diff(cluster_no,perc_diff,'all',algorithm)


output_graphs('k_means')
output_graphs('wta')
output_graphs('kohonen')
```

A.4 k-means Algorithm File

```
import numpy as np

def initialize(clusters): #initialize the clusters with empty totals
    clusters['red_total']=np.zeros(shape=len(clusters),dtype=int)
    clusters['green_total']=np.zeros(shape=len(clusters),dtype=int)
    clusters['blue_total']=np.zeros(shape=len(clusters),dtype=int)
    clusters['n_total']=np.zeros(shape=len(clusters),dtype=int)
    clusters['red_mean']=np.nan
    clusters['green_mean']=np.nan
    clusters['blue_mean']=np.nan
    return clusters

def reset_clusters(clusters): #reset clusters to 0 total
    clusters['red_total']=0
    clusters['green_total']=0
    clusters['blue_total']=0
    clusters['n_total']=0
    clusters['red_mean']=0
    clusters['green_mean']=0
    clusters['blue_mean']=0
    return clusters

def sample_mean(pixel_data,clusters): #Finds the rgb sample mean
    for i in range(len(pixel_data)):
        cid=int(pixel_data.closest_id.iloc[i])
        clusters.red_total.iloc[cid]+=pixel_data.red.iloc[i]
        clusters.green_total.iloc[cid]+=pixel_data.green.iloc[i]
        clusters.blue_total.iloc[cid]+=pixel_data.blue.iloc[i]
        clusters.n_total.iloc[cid]+=1

    clusters['red_mean']=clusters['red_total']/clusters['n_total']
    clusters['green_mean']=clusters['green_total']/clusters['n_total']
    clusters['blue_mean']=clusters['blue_total']/clusters['n_total']
    return clusters

def eliminate_empty_clusters(clusters): #removes the clusters with no local pixel_data points
    clusters = clusters[clusters.n_total != 0]
    return clusters

def reassign_cluster_pos(clusters): #reassigns the position of clusters to their means
    clusters['red']=clusters['red_mean']
    clusters['green']=clusters['green_mean']
    clusters['blue']=clusters['blue_mean']
    return clusters

def check_clusters_size_limit(clusters,limit): #checks the number of clusters left, returns
    true if required cluster size is reached
    if len(clusters)<=limit:
        return True
    else:
        return False
```

A.5 WTA and Kohonen Algorithm File

```
import numpy as np

def initialize(clusters): #initialize the clusters with empty totals
    clusters['red_temp']=np.zeros(shape=len(clusters),dtype=int)
    clusters['green_temp']=np.zeros(shape=len(clusters),dtype=int)
    clusters['blue_temp']=np.zeros(shape=len(clusters),dtype=int)
    clusters['phi']=np.zeros(shape=len(clusters))
    return clusters

def reset_clusters(clusters): #reset clusters to 0 total
    clusters['red_temp']=0
    clusters['green_temp']=0
    clusters['blue_temp']=0
    return clusters

def pixel_shift(pixel_data,clusters,epsilon): #shifts the cluster position
    for i in range(len(pixel_data)):
        red_shift=epsilon*(pixel_data['red']-clusters['red'])
        green_shift=epsilon*(pixel_data['green']-clusters['green'])
        blue_shift=epsilon*(pixel_data['blue']-clusters['blue'])
        clusters['red']=clusters['red']+red_shift
        clusters['green']=clusters['green']+green_shift
        clusters['blue']=clusters['blue']+blue_shift
    return clusters


def kohonen(pixel_data,clusters,epsilon_max,epsilon_min,k,k_max):
    epsilon=epsilon_max*(epsilon_min/epsilon_max)**(k/k_max)

    for i in range(len(pixel_data)):
        phi=0
        std_dev=0
        cid=pixel_data.closest_id.iloc[i]

        for j in range (0,len(clusters)):
            std_dev_r=((clusters['red']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)
            std_dev_g=((clusters['green']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)
            std_dev_b=((clusters['blue']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)

            a=(clusters.red.iloc[cid]-clusters.red.iloc[j])**2+(clusters.green.iloc[cid]-
    clusters.green.iloc[j])**2+(clusters.blue.iloc[cid]-clusters.blue.iloc[j])**2
            std_dev=(std_dev_r+std_dev_g+std_dev_b)/3
            phi=np.exp(a/(2*std_dev)) #calculated phi value for shift
            clusters.phi.iloc[j]=phi

        red_shift=clusters['phi']*epsilon*(pixel_data['red']-clusters['red'])
        green_shift=clusters['phi']*epsilon*(pixel_data['green']-clusters['green'])
        blue_shift=clusters['phi']*epsilon*(pixel_data['blue']-clusters['blue'])
        clusters['red']=clusters['red']+red_shift
        clusters['green']=clusters['green']+green_shift
        clusters['blue']=clusters['blue']+blue_shift
    return clusters






def kohonen1234(pixel_data,clusters,epsilon_max,epsilon_min,k,k_max): #shifts the cluster
    position
    epsilon=epsilon_max*(epsilon_min/epsilon_max)**(k/k_max)
    red_shift=(pixel_data['red']-clusters['red'])
    green_shift=(pixel_data['green']-clusters['green'])
    blue_shift=(pixel_data['blue']-clusters['blue'])

    for j in range (0,len(clusters)):
        std_dev_r=((clusters['red']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)
        std_dev_g=((clusters['green']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)
        std_dev_b=((clusters['blue']-clusters.red.iloc[j])**2).sum()/(len(clusters)-1)
        std_dev=(std_dev_r+std_dev_g+std_dev_b)/3
        for i in range(0,len(clusters)):
            print(clusters)
            a=(clusters.red.iloc[i]-clusters.red.iloc[j])**2+(clusters.green.iloc[i]-clusters.
    green.iloc[j])**2+(clusters.blue.iloc[i]-clusters.blue.iloc[j])**2
            print('red',clusters.red.iloc[i]-clusters.red.iloc[j])
            print('green',clusters.green.iloc[i]-clusters.green.iloc[j])
            print('blue',clusters.blue.iloc[i]-clusters.blue.iloc[j])

            print(a)
            print(std_dev)

            phi=np.exp(a/(2*std_dev))
            clusters.red_temp.iloc[i]=red_shift
            clusters.red_temp.iloc[i]=phi
            clusters.red_temp.iloc[i]=epsilon
            clusters.red_temp.iloc[i]=clusters.red.iloc[i]+red_shiftclusters.red.iloc[i]*phi*
    epsilon
            clusters.green_temp.iloc[i]=clusters.green.iloc[i]+green_shift*phi*epsilon
            clusters.blue_temp.iloc[i]=clusters.blue.iloc[i]+blue_shift*phi*epsilon

    clusters['red']=clusters['red_temp']
    clusters['green']=clusters['green_temp']
    clusters['blue']=clusters['blue_temp']
    return clusters
```

A.6 Input/Output Image File

```
#reads in picture and outputs rgb and clusters with closest cluster for each data point
#outputs clusters.csv for cluster locations
#outputs pixel_data.csv for each data point with rgb and closest cluster and closest cluster
    dist

import sys
import math
import timeit
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn import svm
from skimage import io
from PIL import Image

def read_in_fig(figure):
    image = io.imread(figure)
    io.imshow(image)
    io.show()
    rows = image.shape[0]
    cols = image.shape[1]

    #reads in rgb list from image
    r=[]#length of each should be 120*120
    g=[]
    b=[]
    c=[]
    cdist=[]
    for line in image:
        for pixel in line:
            temp_r,temp_g,temp_b=pixel
            r.append(temp_r)
            g.append(temp_g)
            b.append(temp_b)
            c.append(np.NaN)
            cdist.append(np.NaN)
    data = pd.DataFrame({'red': r,'green': g,'blue': b,'closest_id':c,'closest_dist':cdist})
    return data

def read_out_fig(pixel_data,image_input_name,name):#reads out image to same size as read in
    image in image_name
    image = io.imread(image_input_name)
    rows = image.shape[0]
    cols = image.shape[1]
    data = np.zeros( (rows,cols,3), dtype=np.uint8 )
    count=0

    for row in range(0,rows):
        for column in range(0,cols):
            data[row,column] = [pixel_data.r_cluster.iloc[count],pixel_data.g_cluster.iloc[
    count],pixel_data.b_cluster.iloc[count]]  #inputs pixel color for specific pixel
            count+=1
    img = Image.fromarray( data )
    img.save('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/images/'+name+'.png')


def initialize_clusters(n): #initialize clusters
    cluster = pd.DataFrame(index=range(0,n),columns=['red','green','blue'])
    cluster['red']=np.linspace(0,256,n)
    cluster['green']=np.linspace(0,256,n)
    cluster['blue']=np.linspace(0,256,n)
    return cluster
```

A.7 Output Data File

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_rgb_cluster_pix(cluster,pixel,name): #plots rgb for cluster and pixels
    cluster_fig = plt.figure()
    ax=Axes3D(cluster_fig)
    ax.scatter(cluster['red'],cluster['green'],cluster['blue'],color='red',s=10)
    ax.scatter(pixel['red'],pixel['green'],pixel['blue'],color='blue',s=10)
    ax.set_xlabel('red')
    ax.set_ylabel('green')
    ax.set_zlabel('blue')
    plt.savefig('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/plots/'+name+'.png'
    )
    print('rgb plot saved with cluster and pixels')

def save_file(cluster,pixel_data,name):
    cluster.to_csv('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/data/'+name+'
    _clusters.csv',index=False)
    pixel_data.to_csv('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/data/'+name+'
    _pixel_data.csv',index=False)

def plot_rgb_cluster(cluster,pixel,name): #plots rgb for cluster and pixels
    cluster_fig = plt.figure()
    ax=Axes3D(cluster_fig)
    ax.scatter(cluster['red'],cluster['green'],cluster['blue'],color='red',s=10)
    ax.set_xlabel('red')
    ax.set_ylabel('green')
    ax.set_zlabel('blue')
    plt.savefig('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/plots/'+name+'
    cluster.png')
    print('rgb plot saved with clusters')

def plot_perc_diff(cluster_no,perc_diff,color_name,algorithm):
    plt.figure()
    plt.plot(cluster_no,perc_diff)
    plt.xlabel('Number of clusters')
    plt.ylabel('Percentage difference')
    plt.show()
    plt.savefig('/home/across/UTK_PhD/Machine_learning_fall_2019/project_4/plots/'+algorithm+'
    _'+color_name+'.png')
```

A.8 Find Closest Cluster File

```
import numpy as np
import pandas as pd

def find_coord_closest_cluster(coordinate, cluster): #find closest cluster for single
    coordinate
    coord_comp=(coordinate['red']-cluster['red'])**2+(coordinate['green']-cluster['green'])
    **2+(coordinate['blue']-cluster['blue'])**2
    minid=int(coord_comp.idxmin())
    mindist=np.sqrt(coord_comp.min())

    return minid,mindist

def find_all_closest_cluster(data_pixel,cluster): #find closest cluster for all coordinates
    for i in range(len(data_pixel)):
        data_pixel.closest_id.iloc[i], data_pixel.closest_dist.iloc[i]=
    find_coord_closest_cluster(data_pixel.iloc[i],cluster.copy())

    return data_pixel

def update_pixels_to_closest_cluster(pixel_data,clusters): #updates pixel to the rgb value of
    closest cluster
    pixel_data['closest_id']=pixel_data['closest_id'].astype(int)
    r=clusters.red.iloc[pixel_data['closest_id']].copy()
    g=clusters.green.iloc[pixel_data['closest_id']].copy()
    b=clusters.blue.iloc[pixel_data['closest_id']].copy()
    r=r.tolist()
    g=g.tolist()
    b=b.tolist()
    cluster_addition = pd.DataFrame(list(zip(r, g,b)), columns =['r_cluster', 'g_cluster','
    b_cluster'])
    pixel_data=pd.concat([pixel_data,cluster_addition],axis=1)
    return pixel_data

def reset_closest_cluster(data_pixel):
    data_pixel['closest_id']=0
    data_pixel['closest_dist']=0

    return data_pixel
```

References

Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

University of Tennessee. Machine learning cosc 522 lecture 15. `http://web.eecs.utk.edu/~hqi/cosc522/`. Accessed: 11/23/2019.