Kunal Goyal
CS 558
HW4
Computer Vision

Code:

Main.py:

```
"""
Name: Kunal Goyal
CS558
Computer Vision
"""
import numpy as np
import random
import itertools
import math
import cv2
import slic
import kmeans
import pandas as pd
import matplotlib.pyplot as plt
from tkinter import *
from tkinter import messagebox
from PIL import ImageTk, Image
from tkinter import filedialog

def kmeansStart():

    original_image = cv2.imread("white-tower.png")
    img=cv2.cvtColor(original_image,cv2.COLOR_BGR2RGB)
    vectorized = img.reshape((-1,3))
    vectorized = np.float32(vectorized)

    data=pd.DataFrame(vectorized,columns=['R','G','B'])
    k= 10

    [x,y,z,a,b]= kmeans.k_mean(data,k)

    m=y.index.tolist()
    n=x.index.tolist()
    o=z.index.tolist()


    list_=[]
    for i in range (len(m)):
        h1=(y.loc[m[i]]).tolist()
        h2=x[x['class']==i+1]
        l1=len(h2)
        for j in range (l1):
            list_.append(h1)


    res1=pd.DataFrame(list_,columns=['R','G','B'],index=n)
    res1=res1.reindex(o)
    list_1=[]
    for g in range (len(res1)):
        h3=(res1.loc[g]).tolist()
        list_1.append(h3)
    listx=np.array(list_1)
    listx=np.uint8(listx)
    result_image = listx.reshape((img.shape))
    #cv2.imwrite("white-tower-kmeans.png",result_image)
```

Kunal Goyal
CS 558
HW4
Computer Vision

```
    figure_size = 15
    plt.figure(figsize=(figure_size,figure_size))
    plt.subplot(1,2,1),plt.imshow(img)
    plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    plt.subplot(1,2,2),plt.imshow(result_image)
    h5=str(k)
    plt.title('image for k='+h5), plt.xticks([]), plt.yticks([])
    plt.savefig('white-tower-kmeansplt.png')
    plt.close('all')


def slicStart():
    img = cv2.imread("wt_slic.png")
    img = cv2.resize(img,(int(img.shape[1]*0.5),int(img.shape[0]*0.5)),interpolation = cv2.INTER_AREA)
    slic.impSlic(img)

if __name__ == "__main__":

    try:
        kmeansStart()
        slicStart()
    except Exception as e:
        print(e)
```

kmeans.py :

```
import math
import random
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tkinter import *
from tkinter import messagebox
from PIL import ImageTk, Image
from tkinter import filedialog

def k_mean(data,k):
    iters=0
    q=1
    r_mean=data.sample(k)
    while q>0:
        ind=r_mean.index.tolist()
        indx=data.index.tolist()
        col=data.columns.tolist()
        dist=pd.DataFrame()
        for i in range (k):
            distance=(data[col] - np.array(r_mean.loc[ind[i]])).pow(2).sum(1).pow(0.5)
            v=i+1
            v=str(v)
            dist['d'+v]=distance
        col_dist=dist.columns.tolist()

        var=pd.DataFrame(dist.idxmin(axis=1),columns=['class'])
        u_mean=[]
        col.append('class')
        new_data=pd.DataFrame(columns=col)
        for i in range(k):
```

```
        index1=(var[var['class']==col_dist[i]]).index.tolist()
        cluster=data.loc[data.index.isin(index1)]
        m=cluster.mean().tolist()
        u_mean.append(m)
        l=len(cluster)
        list1=[]
        for j in range (l):
            list1.append(i+1)
        cluster['class']=list1
        new_data=new_data.append(cluster)
    col.remove(col[-1])
    u_mean=pd.DataFrame(u_mean,columns=col,index=ind)
    if u_mean.equals(r_mean):
        q=0
    else:
        r_mean=u_mean

    res=new_data.reindex(indx)
    iters=iters+1
    label=res['class'].tolist()
    temp1=set(label)
    temp1=list(temp1)
    var3=0
    for i in range (k):
        for j in range (len(label)):
            var2=label[j]
            if var2==temp1[i]:
                label[j]=var3
        var3=var3+1
  return new_data,u_mean,res,iters,label
```

slic.py:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
from skimage import color
from tqdm import tqdm
from PIL import Image
import PIL
from tkinter import *




def impSlic(img):
  lab = color.rgb2lab(img)
  N = img.shape[0]*img.shape[1]
  K = 50
  S = math.sqrt(N/K)

  # Creating the grid with S as spacing
  cod = []
  frame = np.zeros((img.shape[0],img.shape[1]))
  for x in range(int(S),img.shape[0],int(S)):
    for y in range(int(S),img.shape[1],int(S)):
      cod.append([x,y])
```

Kunal Goyal
CS 558
HW4
Computer Vision

```
        frame[x,y] = 255

    #Calculating the gradient image
    grad = np.zeros((img.shape[0],img.shape[1]))
    figure=plt.figure(figsize=(10,10))
    plt.subplot(2,2,1)
    plt.imshow(np.uint8(lab))
    plt.title('Image in ceilab mapping')

    for x in range(1,img.shape[0]-1):
        for y in range(1,img.shape[1]-1):
            X = []
            Y = []
            for i in range(3):
                xx = lab[x+1,y,i] - lab[x-1,y,i]
                yy = lab[x,y-1,i] - lab[x,y+1,i]
                X.append(xx)
                Y.append(yy)

            grad[x,y] = np.linalg.norm(X) + np.linalg.norm(Y)
    plt.subplot(2,2,2)
    plt.imshow(np.uint8(grad),cmap = 'gray')
    plt.title('Gradient')
    dst = cv2.addWeighted(grad, 0.4, frame, 0.6, 0.0)
    plt.subplot(2,2,3)
    plt.imshow(dst,cmap = 'gray')
    plt.title('Gradient with initial means')
    for i in range(len(cod)):
        pp = cod[i]
        q = grad[pp[0]-4:pp[0]+4,pp[1]-4:pp[1]+4]
        z = np.amin(q)
        result = np.where(q==z)
        t = list(zip(result[0],result[1]))[0]
        cod[i] = [pp[0]-4 + t[0],pp[1]-4 + t[1]]

    frame2 = np.zeros((img.shape[0],img.shape[1]))

    for loc in cod:
        x = loc[0]
        y = loc[1]
        frame2[x,y] = 255
    dst2 = cv2.addWeighted(grad, 0.4, frame2, 0.6, 0.0)
    plt.subplot(2,2,4)
    plt.imshow(dst2,cmap = 'gray')
    plt.title('Gradient with means without edges')

    newImg = np.zeros(img.shape)
    indexImg = np.zeros((img.shape[0],img.shape[1]))
    for iters in tqdm(range(10)):
        for x in range(img.shape[0]):
            for y in range(img.shape[1]):

                #Get the mean point with distance less than 2*S
                temp = []
                for i in range(len(cod)):
                    dist = math.sqrt((x-cod[i][0])**2 + (y-cod[i][1])**2)
                    if dist <= 2*S:
                        temp.append(cod[i])

                td = []
```

Kunal Goyal
CS 558
HW4
Computer Vision

```
            for j in range(len(temp)):
                Dlab = math.sqrt((lab[:,:,0][x,y]-lab[:,:,0][temp[j][0],temp[j][1]])**2 + (lab[:,:,1][x,y]-
lab[:,:,1][temp[j][0],temp[j][1]])**2 + (lab[:,:,2][x,y]-lab[:,:,2][temp[j][0],temp[j][1]])**2)
                Dxy  = math.sqrt((x-temp[j][0])**2 + (y-temp[j][1])**2)
                dist = Dlab - (20/S)*Dxy
                td.append(dist)
            if len(td)!=0:
                cord = temp[np.argmin(td)]

            newImg[x,y,0] = img[cord[0],cord[1],0]
            newImg[x,y,1] = img[cord[0],cord[1],1]
            newImg[x,y,2] = img[cord[0],cord[1],2]
            r = 0
            for item in cod:
                if list(item) == list(cord):
                    indexImg[x,y] = r
                r+=1

    for k in range(len(cod)):
        result1 = np.where(indexImg==k)
        if len(result1[0])!=0:
            cod[k] = np.array([int(np.mean(result1[0])),int(np.mean(result1[1]))])

    frame1 = np.zeros((img.shape[0],img.shape[1]))
    for l in range(len(cod)):
        frame1[cod[l][0],cod[l][1]] = 255
    plt.figure()
    plt.imshow(np.uint8(newImg))
    j = Image.fromarray(np.uint8(newImg), mode='RGB')

cv2.imshow('final',np.uint8(newImg))
cv2.imwrite("slicImage.png", np.uint8(newImg))
```

OutputImage:

Original Image

image for k=10

Kunal Goyal
CS 558
HW4
Computer Vision

Kmeans for k=10

Slic on white-tower.png

Slic on wt-slic.png

Kunal Goyal
CS 558
HW4
Computer Vision

Kmeans:

K-means is similar to KNN because it looks at distance to predict class membership. However, unlike KNN, K-means is an unsupervised learning algorithm. Its goal is to discover how different points cluster together. The intuition behind this mathematical model is that similar data points will be closer together.
K-means then tries to determine different k-points called centroids, which are at the centre (least cumulative distance) from other points of the same class, but further away from points of another class.
K-means clustering uses "centroids", K different randomly-initiated points in the data, and assigns every data point to the nearest centroid. After every point has been assigned, the centroid is moved to the average of all of the points assigned to it. Then the process repeats: every point is assigned to its nearest centroid, centroids are moved to the average of points assigned to it. The algorithm is done when no point changes assigned centroid.
The algorithm looks a little bit like…

Initialize K random centroids.
You could pick K random data points and make those your starting points.
Otherwise, you pick K random values for each variable.
For every data point, look at which centroid is nearest to it.
Using some sort of measurement like Euclidean or Cosine distance.
Assign the data point to the nearest centroid.
For every centroid, move the centroid to the average of the points assigned to that centroid.
Repeat the last three steps until the centroid assignment no longer changes.
The algorithm is said to have "converged" once there are no more changes.
These centroids act as the average representation of the points that are assigned to it. This gives you a story almost right away. You can compare the centroid values and tell if one cluster favors a group of variables or if the clusters have logical groupings of key variables.

SLIC:

Simple Linear Iterative Clustering (SLIC) is one of the most excellent superpixel segmentation algorithms with the most comprehensive performance and is widely used in various scenes of production and living. As a preprocessing step in image processing, superpixel segmentation should meet various demands in real life as much as possible, but SLIC is highly sensitive to noise.
Simple Linear Iterative Clustering is the state-of-the-art algorithm to segment superpixels which doesn't require much computational power. In brief, the algorithm clusters pixels in the combined five-dimensional color and image plane space to efficiently generate compact, nearly uniform superpixels
The approach is simple. SLIC performs a local clustering of pixels in 5-D space defined by the L, a, b values of the CIELAB color space and x, y coordinates of the pixels. It has a different distance measurement which enables compactness and regularity in the superpixel shapes and can be used on grayscale images as well as color images.

Kunal Goyal
CS 558
HW4
Computer Vision

SLIC generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. A 5 dimensional [labxy] space is used for clustering. CIELAB color space is considered as perpetually uniform for small color distances. It is not advisable to simply use Euclidean distance in the 5D space and hence the authors have introduced a new distance measure that considers superpixels size.