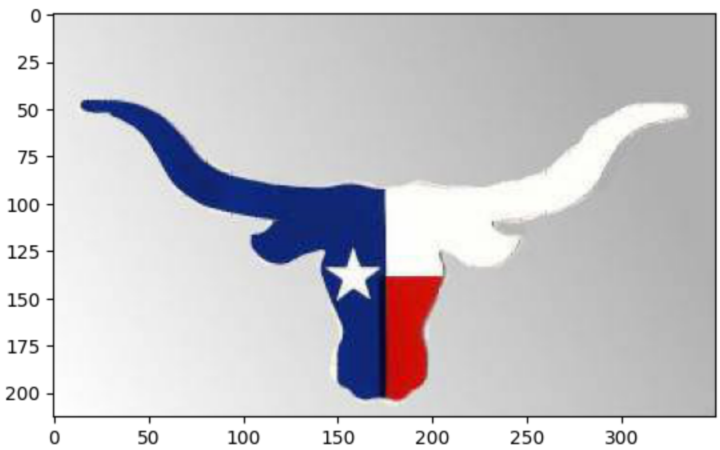
```
#Load libraries
import matplotlib.image as mpimg
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.sparse import csr_matrix
from scipy.sparse import diags
from scipy import sparse as sp
import pickle
```

Part a

```
# Read Images
pkl_file = open('/content/AUSTIN_TEXAS_Hw10.pkl', 'rb')
img = pickle.load(pkl_file)
pkl_file.close()

###############################################################################
# Parts a and b
### INSERT YOUR CODE HERE ###
plt.imshow(img)
print(img.shape)
```

```
(213, 350, 3)
```



Answer b)

What are the dimensions of the image? **The dimensions of the image is (213, 350, 3)**

Part c

```
###############################################################################
# Setup for part c and beyond
I = img.shape[0]
J = img.shape[1]
numPixels = I * J

redPixelVals = img[:,:,0].flatten()
greenPixelVals = img[:,:,1].flatten()
bluePixelVals = img[:,:,2].flatten()
###############################################################################
# Part c
#Create Queens Neighbor Matrix
#CSR Sparse Initialization
neighborMat = csr_matrix((numPixels,numPixels))

#By Definition, the neighbors of a pixel are the following pixels
adjToNeighbor = np.array([1,J-1,J,J+1,-1,-(J-1),-J,-(J+1)])
neighborGrid = np.array([x + adjToNeighbor for x in range(numPixels)])

#Now we need to adjust the matrix to remove issues at the boundaries
rowNums = np.asarray(np.repeat(list(range(numPixels)),8)).reshape([numPixels,8])
neighborGrid[neighborGrid < 0] = rowNums[neighborGrid < 0]
neighborGrid[neighborGrid > numPixels-1] = rowNums[neighborGrid > numPixels-1]
neighborGrid[J * np.array(range(I)),1] = J * np.array(range(I))
neighborGrid[J * np.array(range(I)),4] = J * np.array(range(I))
neighborGrid[J * np.array(range(I)),7] = J * np.array(range(I))
neighborGrid[J * np.array(range(1,I+1))-1,5] = J * np.array(range(1,I+1))-1
neighborGrid[J * np.array(range(1,I+1))-1,3] = J * np.array(range(1,I+1))-1
```

```
neighborGrid[J * np.array(range(1,I+1))-1,0] = J * np.array(range(1,I+1))-1

#Is there a more direct way of doing all this? Probably. But this all works
neighborMat[np.repeat(list(range(numPixels)),8),neighborGrid.flatten()] = 1
neighborMat[np.array(range(numPixels)),np.array(range(numPixels))] = 0

#Find neighbors of pixel 73850
### INSERT YOUR CODE HERE ###


n_73850=neighborMat[73850,:]

print(n_73850[0,10])


neighboors_indices=[]
for i in range(n_73850.shape[1]):
  if n_73850[0,i]==1:
    neighboors_indices.append(i)
```

```
0.0
```

```
print(neighboors_indices)
```

```
[73500, 73501, 73851, 74200, 74201]
```

```
n_neighboors_indices=np.array(neighboors_indices)

n_neighboors_indices=n_neighboors_indices.astype(np.float)
row=[]
column=[]
for i in range(n_neighboors_indices.shape[0]):
  row.append(n_neighboors_indices[i]//J )
  column.append(n_neighboors_indices[i]-(row[i]*J))
```

```
<ipython-input-42-a3da6a3f101e>:3: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To silence this warni
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  n_neighboors_indices=n_neighboors_indices.astype(np.float)
```

```
print(row)
print(column)
print(J)
```

```
[210.0, 210.0, 211.0, 212.0, 212.0]
[0.0, 1.0, 1.0, 0.0, 1.0]
350
```

```
for i in range(img.shape[0]):
  for j in range(img.shape[1]):
    if i==0 and j==0:
      counter=0
    else :
      counter=counter+1
    if counter==73501:
      index_r=i
      index_c=j
      break
  if counter==73500:
    break

print(index_r)
print(index_c)
```

```
210
1
```

Answer

The pixels neighboors of pixel 73850 are the pixels labelled as 73500, 73501, 73851, 74200, 74201.

Part d

```
#Part d
#Now lets find which spots we need to calculate things for the W matrix
```

```
ixW = sp.find(neighborMat == 1)

#Initialzie W matrix and set sigma parameter
s = 30
#W = csr_matrix((numPixels,numPixels))

#Calculate the W Matrix
### INSERT YOUR CODE HERE ###



rows=ixW[0]
columns=ixW[1]
W=np.zeros((numPixels,numPixels))
for n in range(rows.shape[0]):
    i=rows[n]
    j=columns[n]
    W[i,j]=np.exp(-((redPixelVals[i]-redPixelVals[j])**2+(greenPixelVals[i]-greenPixelVals[j])**2+(bluePixelVals[i]-bluePixelVals[j])**2)/(
```

```
#Calculate the D Matrix
D1 = np.squeeze(np.asarray(np.sum(W,axis = 1)))

#Check the values of the D matrix
print("Minimum,", D1.min())
print("Maximum," ,D1.max())
```

```
Minimum, 0.00015761396890586016
Maximum, 8.0
```

```
#Report values of D matrix
### INSERT YOUR CODE HERE ###
D=np.diag(D1)

print("D[0,0] =",np.round(D[0,0],3))
print("D[612,612] =",np.round(D[612,612],3))
print("D[72630,72630] = ",np.round(D[72630,72630],3))
```

```
D[0,0] = 3.0
D[612,612] = 7.998
D[72630,72630] =  5.45
```

Answer

D[0,0] = 3.0

D[612,612] = 7.998

D[72630,72630] = 5.45

Part e

```
D_sp=sp.csr_matrix(D)
"""
L=D-W
L_SM=np.linalg.inv(D).dot(L)"""
```

```
'\nL=D-W\nL_SM=np.linalg.inv(D).dot(L)'
```

```
print(D_sp.shape)
```

```
(74550, 74550)
```

```
W=sp.csr_matrix(W)
```

```
L = (D_sp.tocsr() - W.tocsr()).tolil()
```

```
D_inv=sp.linalg.inv(D_sp)
```

```
/usr/local/lib/python3.9/dist-packages/scipy/sparse/linalg/_dsolve/linsolve.py:394: SparseEfficiencyWarning: splu converted its input
  warn('splu converted its input to CSC format', SparseEfficiencyWarning)
/usr/local/lib/python3.9/dist-packages/scipy/sparse/linalg/_dsolve/linsolve.py:285: SparseEfficiencyWarning: spsolve is more efficient
  warn('spsolve is more efficient when sparse b '
```

```
L_sm=D_inv.dot(L)


"""count_z=0
for i in range(L.shape[0]):
  for j in range(L.shape[1]):
    if L[i,j]!=0:
      count_z=count_z+1
"""
dff_zero = sp.find(L_sm != 0)

print("Number of elements different of zero",len(dff_zero))


#Part e
#Create the L matrix and LSM matrix
### INSERT YOUR CODE HERE ###
```

```
Number of elements different of zero 3
```

```
print("L_sm[55,62]= ", np.round(L_sm[55,62],3))
print("L_sm[34331,34332]= ", np.round(L_sm[34331,34332],3))
print("L_sm[74199,74548]= ", np.round(L_sm[74199,74548],3))
```

```
L_sm[55,62]=  0.0
L_sm[34331,34332]=  -0.124
L_sm[74199,74548]=  -0.192
```

Answer

The number of elements different of zero are 3.

L_sm[55,62]= 0.0

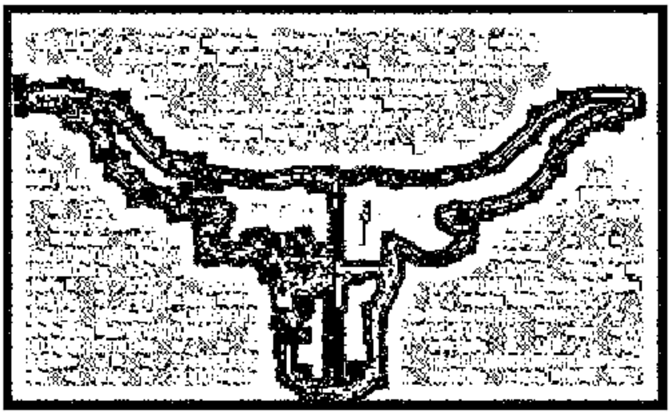
L_sm[34331,34332]= -0.124

L_sm[74199,74548]= -0.192

Part f

```
#Part f
#Find the eigenvectors (and values) of the L matrix
K = 10
M = csr_matrix(diags(D1))
Minv = csr_matrix(diags(1/D1))
#initVec = (np.diag(D) > 6)
initVec = (D_sp.diagonal() > 6)
#Other options for initVec that yield different results
# initVec = (redPixelVals > 100)/numPixels
# initVec = ((redPixelVals + bluePixelVals + greenPixelVals) > 550)/numPixels
initVec = initVec / sum(initVec)
lam, v = sp.linalg.eigs(L_sm,K,M = M,Minv = Minv,v0 = initVec)

#Pull out the top K vectors to use
Y = np.real(v[:,0:K])

#Normalize the rows of Y
Y = np.transpose(np.transpose(Y)/np.linalg.norm(Y,axis = 1))
Y = np.nan_to_num(Y)

#Perform K-Means on Y
kmeans = KMeans(init = 'k-means++',n_clusters = 5,n_init = 10,
                max_iter = 300,random_state=0)
kmeans.fit(Y)

#Create a plot
from matplotlib.colors import from_levels_and_colors
cmap , norm = from_levels_and_colors([-.5,0.5,1.5], ['white','black'])

pred=kmeans.predict(Y)
clusterImg=np.reshape(pred,(img.shape[0],img.shape[1]))
plt.imshow(clusterImg,cmap=cmap)
ax = plt.gca()
```

```
ax.axes.xaxis.set_visible(False)
ax.axes.yaxis.set_visible(False)
plt.show()
```



Answer

What does the method do well in segmenting? **It does a good job segmenting the bull skull boundary lines from the the background.**

What does the method do poorly in segmenting? **It does a poor job segmenting the star in the bull skull. It does a poor job segmenting the background as just one cluster given that the original picture the gray background has gradient effect (the method introduces clustering as there where noise in the background). The blue part of the skull shouldn't be divided , however the method divides due to the poor segmentation of the star, like it where segmented in two clusters which is not a correct segmentation. In the white part of the bull skull, the method segmented in such way that there is a small cluster, which looks like noise. It shouldn't be there and all white part of the skull should have been segmented as one cluster. Furthermore, in the blue top part of the skull, the method segmented in such a way that there are tiny black clusters. This is noise introduced by the method given that all the blue part should have been clustered as just one cluster. Finally, the boundary lines of the skull have white clusters, as there where different segmentations there, which is noise introduced by the method.**