EE5175 - Lab 9

Otsu and K-Means

Nikilesh B
EE17B112

1. In this assignment we are first going to calculate the global threshold value of the given images palmleaf1.pgm and palmleaf2.pgm using Otsu's thresholding algorithm.

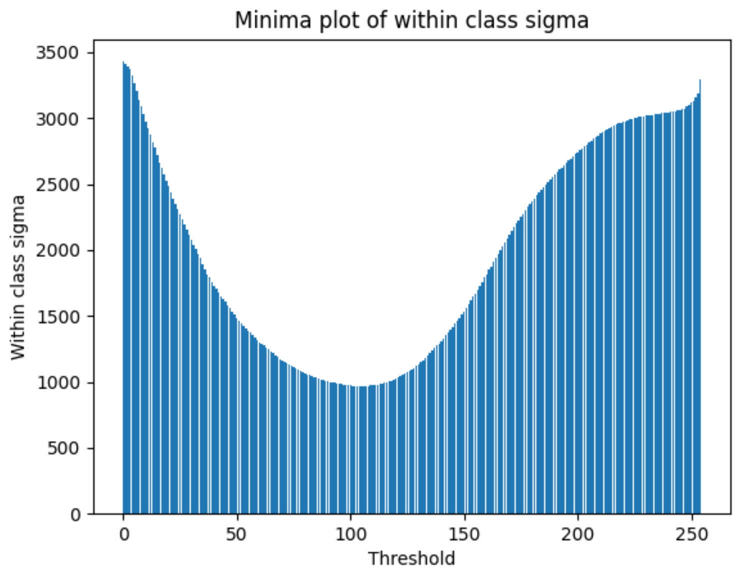
The code for loading the images is as follows :

```
#Load image
src1 = cv2.imread("palmleaf1.pgm",0)
src2 = cv2.imread("palmleaf2.pgm",0)
```

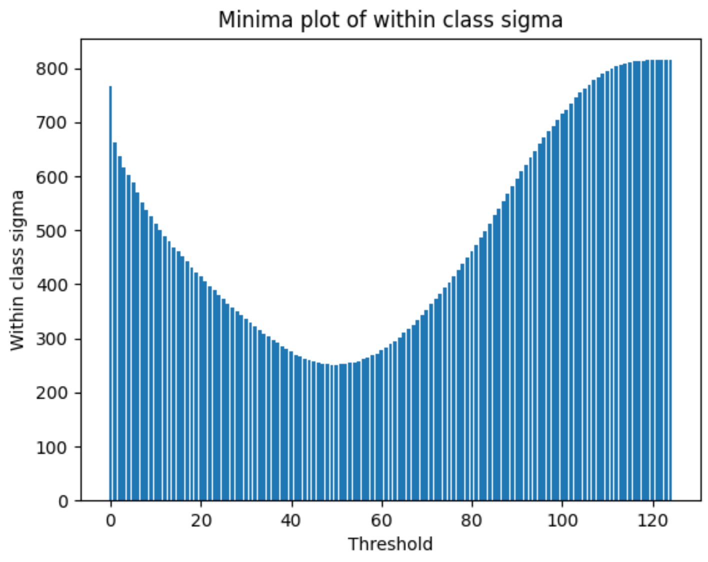
The code for calculating the global threshold value is as follows :

```
#Find global threshold
def gthres(F,L) :
      nL = np.arange(0,L)
      N = sum(F)
      muT = (sum(nL*F))/N
      sigwin = [] ; siginb = []
      for t in range(0,L) : #Find parameters
      N1 = sum(F[:t+1])  ;  N2 = sum(F[t+1:])
      mu1 = (sum(nL[:t+1]*F[:t+1]))/N1 ; mu2 = (sum(nL[t+1:]*F[t+1:]))/N2
      sig1 = (sum(((nL[:t+1]-mu1)**2)*F[:t+1]))/N1
      sig2 = (sum(((nL[t+1:]-mu2)**2)*F[t+1:]))/N2
      sigval1 = (sig1*N1 + sig2*N2)/N
      sigwin.append(sigval1)
      sigval2 = ((mu1-muT)*(mu1-muT)*N1 + (mu2-muT)*(mu2-muT)*N2)/N
      siginb.append(sigval2)
      gthres = np.where(sigwin == min(sigwin))[0]

      plt.bar(nL,sigwin)      #Plot the sigma values
      plt.title("Minima plot of within class sigma")
      plt.xlabel("Threshold")
      plt.ylabel("Within class sigma")
      plt.show()
```

- For palmleaf1.pgm

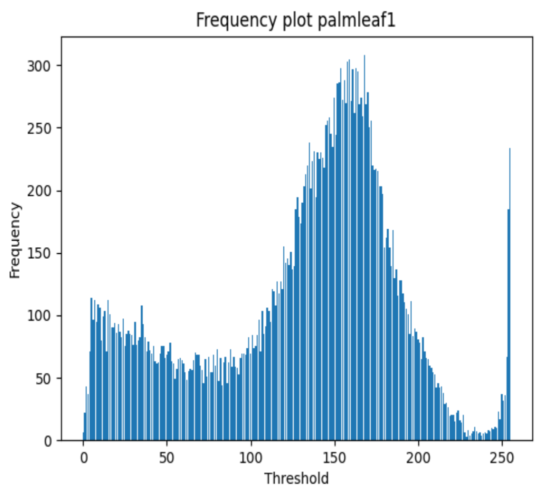


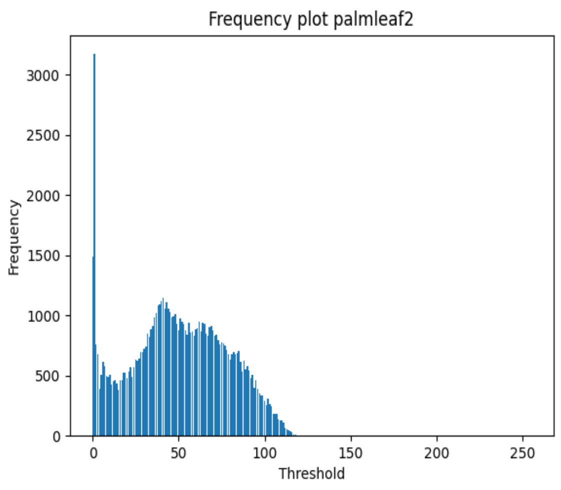
- For palmleaf2.pgm



The code for obtaining the frequency plots and the obtained plots are as follows :

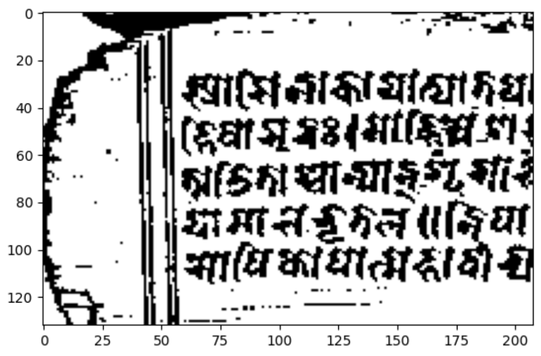
```
#Obtain intensity frequency
L = 256
F1 = []
F2 = []
for i in range(0,L) :
      ind1 = np.where(src1 == i)[0]
      ind2 = np.where(src2 == i)[0]
      F1.append(len(ind1))
      F2.append(len(ind2))

F1 = np.array(F1)
F2 = np.array(F2)
#Frequency plots
plt.bar(range(0,L),F1)
plt.title("Frequency plot palmleaf1")
plt.xlabel("Threshold")
plt.ylabel("Frequency")
plt.show()
plt.bar(range(0,L),F2)
plt.title("Frequency plot palmleaf2")
plt.xlabel("Threshold")
plt.ylabel("Frequency")
plt.show()
```



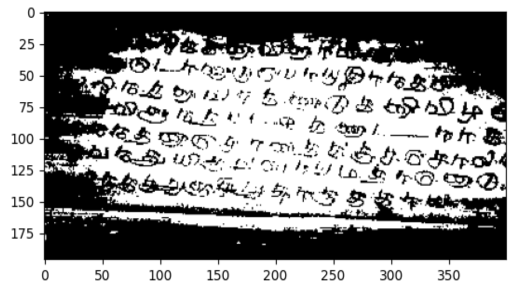


The code for displaying the threshold binary images is as follows :

```
#Thresholded pictures
t1 = gthres(F1,L)
t2 = gthres(F2,L)
print(t1,t2)

thresimg1 = np.zeros(src1.shape)
thresimg2 = np.zeros(src2.shape)

ind1 = np.where(src1 <= t1)
ind2 = np.where(src1 > t1)
thresimg1[ind1] = 0
thresimg1[ind2] = 1

ind1 = np.where(src2 <= t2)
ind2 = np.where(src2 > t2)
thresimg2[ind1] = 0
thresimg2[ind2] = 1

plt.imshow(thresimg1,cmap = "gray")
plt.show()
plt.imshow(thresimg2,cmap = "gray")
plt.show()
```



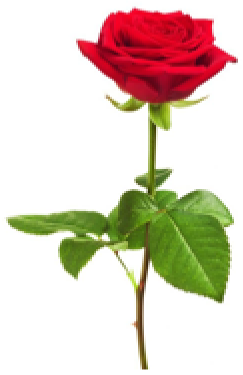
Palmleaf1

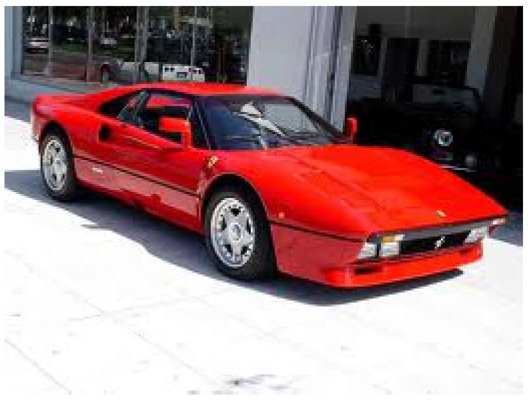


Palmleaf2

2. In the following question, we perform K-means clustering on the two input images (car.ppm and flower.png) for K = 3 clusters.
We use only Euclidean distance as the distance measure for all iterations. We perform 5 iterations of the algorithm. To visualize the output of k-means clustering, we replace each pixels in the input image with the cluster center it belongs to and display the resulting Image.

The **input images** are as follows :





(a) We perform K-means clustering with initial cluster means as follows:

cinit1 - [255 0 0]
cinit2 - [0 0 0]
cinit3 - [255 255 255]

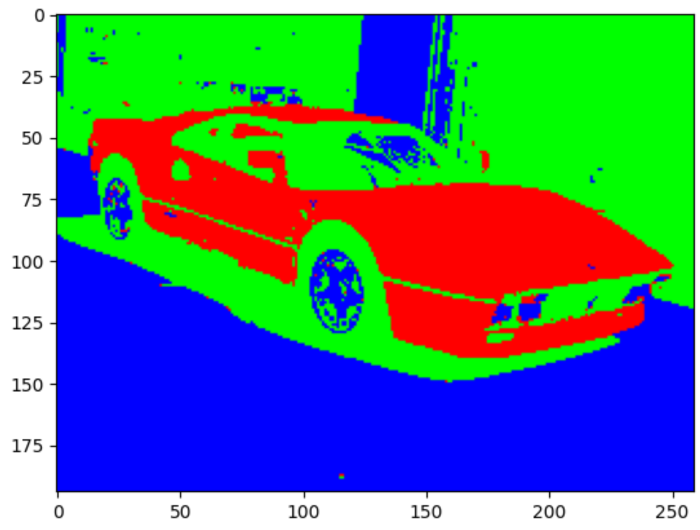
The code for reading the images and initializing cluster means is as follows :

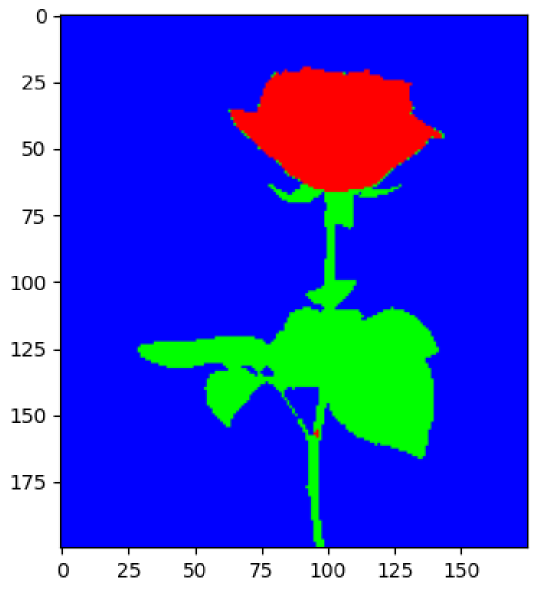
```
src1 = cv2.imread("flower.png")
src2 = cv2.imread("car.ppm")

#Parameters
c1 = np.array([0,255,0])
c2 = np.array([0,0,0])
c3 = np.array([255,255,255])
C = np.array([c1,c2,c3])
```

The code for the k-means function block is as follows:

```
def kmeans(img,K,C,Iter=5) :

      #Divide the dataset into K-clusters
      imgK = np.zeros((K,img.shape[0],img.shape[1]))
      newC = []
      ssq = 0
      indLbl = []
      for k in range(0,K) :         #Norm with respect to centres
      imgK[k] = norm(img-C[k],axis=2)
      label = np.argmin(imgK,axis=0)      #Fix the labels
      #print("0",len(np.where(label==0)[0]))
      #print("1",len(np.where(label==1)[0]))
      #print("2",len(np.where(label==2)[0]))

      #Recompute the pattern centres
      for k in range(0,K) :
      ind = np.where(label==k)
      indLbl.append(ind)
      clstr = img[ind]              #Compute Sum of squares
      ssq = ssq + sum(np.norm(clstr-C[k],axis=1)**2)
      newC.append(clstr.mean(axis=0).astype(int))

      #print(newC)
      #print(len(indLbl))
      newC = np.array(newC)
      if np.array_equal(newC,C) :
      return newC,indLbl,ssq
      #Recursion and check if label are same
      Iter = Iter-1
      if Iter <= 0 :
      return newC,indLbl,ssq
      cntr,indx,ssq = kmeans(img,K,newC,Iter)
      return cntr,indx,ssq
```

The images obtained are attached below :





(b) Now we perform K-means clustering on both images using random initialization of cluster means. We generate 3 random vectors of size 13 that are sampled from uniform distribution in [0 255] and use them as the cluster centers to begin the K-means with. We perform K-means clustering using N such initializations.

The code for that is attached below :

```
#Random initializations
#All centres and ssq
sqsum1 = []
sqsum2 = []
centres = []

for n in range(0,N) :
      #Generate random pixels
      c1 = random.sample(range(0,256),3)
      c2 = random.sample(range(0,256),3)
      c3 = random.sample(range(0,256),3)
      C = np.array([c1,c2,c3])
      centres.append(C)
      C,indLbl,ssq = kmeans(src1,K,C,1)
      sqsum1.append(ssq)
      C,indLbl,ssq = kmeans(src2,K,C,1)
      sqsum2.append(ssq)
```

The obtained images is attached below :